Cristina Silvano · William Fornaciari
Eugenio Villar *Editors*

Multi-objective Design Space Exploration of Multiprocessor SoC Architectures

The MULTICUBE Approach

Springer

VISIT…

Multi-objective Design Space Exploration of Multiprocessor SoC Architectures

Cristina Silvano • William Fornaciari
Eugenio Villar
Editors

Multi-objective Design Space Exploration of Multiprocessor SoC Architectures

The MULTICUBE Approach

Springer

Prof. Cristina Silvano
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Via Ponzio 34/5,
20133 Milano, Italy
e-mail: silvano@elet.polimi.it

Prof. William Fornaciari
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Via Ponzio 34/5,
20133 Milano, Italy
e-mail: fornacia@elet.polimi.it

Prof. Eugenio Villar
Departamento de Tecnología Electrónica e
Ingeniería de Sistemas y Automática (TEISA)
University of Cantabria, Av. Los Castros s/n,
39005 Santander, Spain
e-mail: villar@teisa.unican.es

ISBN 978-1-4419-8836-2 e-ISBN 978-1-4419-8837-9
DOI 10.1007/978-1-4419-8837-9
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011932796

© Springer Science+Business Media, LLC 2011
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

"That is the exploration that awaits you! Not mapping stars and studying nebula, but charting the unknown possibilities of existence."
Q to Captain Jean Luc Picard
Star Trek: The Next Generation, 1994.

Foreword

The objective of the European research programme in Information and Communication Technologies (ICT) is to improve the competitiveness of European industry and enable Europe to shape and master future developments in ICT. ICT is at the very core of the knowledge based society. European research funding has as target to strengthen Europe's scientific and technology base and to ensure European leadership in ICT, help drive and stimulate product, service and process innovation through ICT use and value creation in Europe, and ensure that ICT progress is rapidly transformed into benefits for Europe's citizens, businesses, industry and governments.

Over the last years, the European Commission has constantly increased the amount of funding going to research in computing architectures and tools with special emphasis on multicore computing. Typically, European research funding in a new area (like multi/many core computing) starts with funding for a Network of Excellence. Networks of Excellence are an instrument to overcome the fragmentation of the European research landscape in a given area by bringing together around a common research agenda the leading universities and research centers in Europe; their purpose is to reach a durable restructuring/shaping and integration of efforts and institutions.

In the following years, a number of collaborative research projects may also be funded to address specific, more industrially oriented, research challenges in the same research area. It is important to note here that collaborative research projects are the major route of funding in the European research landscape in a way that is quite unique worldwide. In European collaborative research projects, international consortia consisting of universities, companies and research centers, are working together to advance the state of the art in a given area. The typical duration of such a project is three years.

In 2004 the European Commission launched the HiPEAC Network of Excellence. In 2006, the European Commission launched the Future and Emerging Technologies initiative in Advanced Computing Architectures as well as a number of projects covering Embedded Computing. In 2008, a new set of projects were launched to address the challenges of the multi/many core transition—in embedded, mobile and general-purpose computing—under the research headings "Computing Systems" and "Embedded Systems". These projects were complemented by a second wave of projects that have started in 2010 under the same research headings together

with a new Future and Emerging Technologies initiative on "Concurrent Tera-device Computing". This effort continues in 2011 with two Calls for Proposals: one under the heading "Computing Systems" with 45 million euro funding and the other under the heading "Exascale Computing" with 25 million euro funding.

The MULTICUBE collaborative research project was funded to perform research on multi-objective design space exploration of multicore architectures targeting embedded applications. Results from MULTICUBE are presented in this book providing a valuable reference point to researchers and engineers.

It has been a long way, but we now have an important computing research community in Europe, both from industry and academia, engaging in collaborative research projects that bring together strong European teams in cutting-edge technologies. The book that you have in your hands is a clear demonstration of the breakthroughs that can be obtained through European collaboration.

Dr. Panagiotis Tsarchopoulos
Project Officer
European Commission

Preface

The ever increasing complexity and integration densities of multiprocessor system-on-chip (MPSoC) architectures, significantly enlarges the design space of embedded computing systems. A wide range of design choices must be tuned from a multi-objective perspective, mainly in terms of performance and energy consumption, to find the most suitable system configuration for the target application. Given the huge design space to be analysed, the exploration of tradeoffs between multiple competing objectives cannot be anymore driven by a manual optimisation process based on the intuition and past experience of the system architect. Multi-objective exploration of the huge design space of next generation multi/many core architectures needs for Automatic Design Space Exploration techniques to systematically explore the design choices and to compare them in terms of multiple competing objectives (trade-offs analysis). Ideally, a designer would try all possible design choices and choose the most suitable according to the specific system requirements. Unfortunately, such an exhaustive approach is often unfeasible because of the large number of design choices to be simulated and analysed, in some cases showing some sophisticated effects on system properties that rarely enable to easily and accurately model the system behavior. Consequently, good search techniques are needed not only to find design alternatives that best meet system constraints and cost criteria, but also to prune the search space to crucial parameters to enable an effective and efficient design space exploration.

In the age of multi/many core architectures, system optimization and exploration definitely represent challenging research tasks. Although many point tools exist to optimize particular aspects of embedded systems, an overall design space exploration framework is needed to combine all the decisions into a global search space with a common interface to the optimization and evaluation tools. The state-of-the art lacks of a Design Space Exploration (DSE) framework to help the designer in the automatic selection of the most suitable system configuration for a certain application given a set of multiple competing objectives.

Based on the idea to provide an automatic DSE framework, we started thinking about a proposal of an European project, namely MULTICUBE[1], that was submitted in May 2007 in the first call of the Seventh Framework Programme on ICT under the Objective 3.3 on Embedded Systems Design. The proposal was accepted and the project started on January 2008 under the coordination of Politecnico di Milano. This book was mainly catalyzed by the main research outcomes and exploitable results of the MULTICUBE European project, where the Editors have acted in 2008–2010 timeframe. The MULTICUBE project focused on the definition of an automatic multi-objective Design Space Exploration (DSE) framework to be used to tune the System-on-Chip architecture for the target application evaluating a set of metrics (e.g. energy, latency, throughput, bandwidth, QoR, etc.) for the next generation embedded multimedia platforms.

The project aimed at increasing the competitiveness of European industries by optimizing the design of embedded computing systems while reducing design time and costs. The project defined an automatic multi-objective DSE framework to be used at design-time to find the best power/performance trade-offs while meeting system-level constraints and speeding up the exploration process. A set of heuristic optimization algorithms have been defined to reduce the exploration time, while a set of response surface modeling techniques have been defined to further speed up the process. Based on the results of the design-time multi-objective exploration, the MULTICUBE project also defined a methodology to be used at run-time to optimize the allocation and scheduling of different application tasks. The design exploration flow results in a Pareto-optimal set of design alternatives in terms of power/performance trade-offs. This set of operating points can then be used at run-time to decide how the system resources should be distributed over different application tasks running on the multiprocessor system on chip.

The MULTICUBE DSE framework leverages a set of open-source and proprietary tools for the exploration, modeling and simulation to guarantee a wide exploitation of the MULTICUBE project results in the embedded system design community. The integration of different tools is ensured by a common XML-based tool interface specification, defined to enable the independent development of modules and a seamless integration of the design tools and the data structures into a common design environment. Several industrial use cases (defined as combination of application and related architecture) have been used to assess the capabilities of the MULTICUBE design flow and tools in an industrial design process. The MULTICUBE project has been strongly industry-driven: industrial partners (STMicroelectronics and DS2) as well IMEC research center have defined the design techniques and tools requirements and then validated them to design some industrial use cases. The benefits of the introduction of the automatic DSE in the design phase of embedded computing systems (justifying its introduction in industrial design processes) have been assessed through a procedure to assess the final objective design quality and the reduction of design turnaround time by introducing such a technology on the entire design

[1] The project acronym, MULTICUBE, stands for: “Multi-Objective design space exploration of multi-processor SoC architectures for embedded multimedia applications”.

process. The benefits on the design process can be measurable and tangible like the reduction of the overall design process lead time, and qualitative or intangible like the streamlining and the reduction of human error prone repetitive operations. The DSE assessment procedure was the basis for the validation of the industrial use cases and demonstrators of the project. Validation results have been assessed based on a set of common assessment criteria.

In the book, we have tried to provide a comprehensive understanding of several facets of the problem of design space exploration for embedded on-chip architectures. The book chapters are organized in two parts. In Part I, several methodologies and tools to support automatic design space exploration are discussed. In Part II of the book, the DSE methodologies and tools described in Part I have then been applied to several application domains to discuss their applicability and to envision their benefits. First, a high-level modeling and exploration approach has been applied for a powerline communication network based on a SoC, then the application of the automatic DSE flow to parallel on-chip architectures is discussed, and finally the DSE for run-time management has been applied to a reconfigurable system for video streaming.

Entering Part I on methodologies and tools, Chap. 1 introduces the MULTICUBE design-flow to support the automatic multi-objective Design Space Exploration (DSE) to tune the parameters of System-on-Chip architectures by considering several metrics such as energy, latency and throughput. One of the important goals of the DSE framework is to find design trade-offs that best meet the system constraints and the cost criteria which are indeed strongly dependent on the target application. The DSE flow is based on the interaction of two frameworks to be used at design time: the Design Space Exploration Framework, a set of open-source and proprietary architectural exploration tools, and the Power/Performance Estimation Framework, a set of modeling and simulation tools (open-source and proprietary) operating at several levels of abstraction. The DSE flow also includes the specification of an XML integration interface to connect the exploration and estimation frameworks and a Run-time Resource Manager that selects, at run-time, the best software configuration alternatives to achieve a good power/performance trade-off.

Chapter 2 introduces M3-SCoPE, an open-source SystemC-based framework for performance modeling of multi-processor embedded systems, software source code behavioral simulation and performance estimation of multi-processor embedded systems. Using M3-SCoPE, the application software running on the different processors of the platform can be simulated efficiently in close interaction with the rest of the platform components. In this way, fast and accurate performance metrics of the system are obtained. These metrics are then delivered to the DSE tools to evaluate the quality of the different configurations in order to select the best power/performance trade-offs.

Chapter 3 presents the optimization algorithms developed in the MULTICUBE project for Design Space Exploration of embedded computing systems. Two software DSE tools implement the optimization algorithms: M3Explorer and modeFRONTIER. The mathematical details of the given optimization problems are explained in

the chapter together with how the algorithms can exchange information with the simulators. The description of the proposed algorithms is the central part of the chapter. The focus is posed on new algorithms and on "ad hoc" modifications implemented in existing techniques to face with discrete and categorical variables, which are the most relevant ones when dealing with embedded systems design. The strategy to test the performance achieved by the optimization is another important topic treated in the chapter. The aim is mainly to build confidence in optimization techniques, rather than to simply compare one algorithm with respect to another one. The "no-free-lunch theorem for optimization" has to be taken into consideration and therefore the analysis will look forward to robustness and industrial reliability of the results. The main contribution of MULTICUBE project in the research field of optimization techniques for embedded systems design is indeed the high level of the obtained compromise between specialization of the algorithms and concrete usability of the DSE tools.

A typical design space exploration flow involves an event-based simulator in the loop, often leading to an actual evaluation time that can exceed practical limits for realistic applications. Chip multi-processor architectures further exacerbate this problem given that the actual simulation speed decreases by increasing the number of cores of the chip. Traditional design space exploration lacks of efficient techniques that reduce the number of architectural alternatives to be analyzed. In Chap. 4, we introduce a set of Response Surface Modeling (RSM) techniques that can be used to predict system level metrics by using closed-form analytical expressions instead of lengthy simulations. The principle of RSM is to exploit a set of simulations generated by one or more Design of Experiments strategies to build a surrogate model to predict the system-level metrics. The response model has the same input and output features of the original simulation based model but offers significant speed-up by leveraging analytical, closed-form functions which are tuned during a model training phase.

Running multiple applications optimally in terms of Quality of Service (e.g., performance and power consumption) on embedded multi-core platforms is a huge challenge. Moreover, current applications exhibit unpredictable changes of the environment and workload conditions which makes the task of running them optimally even more difficult. Chapter 5 presents an automated tool flow which tackles this challenge by a two-step approach: first at design-time, a Design Space Exploration (DSE) tool is coupled with a platform simulator(s) to get optimum operating points for the set of target applications. Secondly, at run-time, a lightweight Run-time Resource Manager (RRM) leverages the design-time DSE results for deciding an operating configuration to be loaded at run-time for each application. This decision is taken dynamically, by considering the available platform resources and the QoS requirements of the specific use-case. To keep RRM execution and resource overhead at minimum, a very fast optimisation heuristic is integrated demonstrating a significant speedup in the optimisation process, while maintaining the desired Quality of Service.

Emerging MPSoC platforms provide the applications with an extended set of physical resources, as well as a well a defined set of power and performance optimization mechanisms (i.e., hardware control knobs). The software stack, meanwhile,

is responsible of taking directly advantage of these resources, in order to meet application functional and non-functional requirements. The support from the Operating System (OS) is of utmost importance, since it gives opportunity to optimize the system as a whole. The main purpose of Chap. 6 is to introduce the reader to the challenges of managing physical and logical resources in complex multi/many-core architectures at the OS level.

Entering Part II on application domains, Chap. 7 presents the application of MULTICUBE methodology to the design of an ITU G.hn compatible component for a powerline communication network based on a SoC. Powerline communication is an advanced telecommunication system enabling fast and reliable transfer of audio, video and data information using the most ubiquitous transmission system: the power lines. This transmission line is used to exchange information between different equipment connected to the network using the advanced coding techniques like such as Orthogonal Frequency Division Multiplexing. The starting point of the analysis is a high level SystemC-based virtual platform for which the chapter analyzes the effects of the variation of a pre-defined set of design parameters on a set of pre-defined metrics. This automatic analysis will drive the design choices in order to build an optimized industrial system. The chapter shows that the SystemC-based virtual platform combined with the MULTICUBE design space exploration framework can save up to 80% of designer's work time, while achieving better results in terms of performance.

Chapter 8 describes two significant applications of the automatic MULTICUBE DSE flow to parallel on-chip architectures. The first part of the chapter presents the design space exploration of a low power processor developed by STMicroelectronics by using the modeFRONTIER tool to demonstrate the benefits DSE not only in terms of objective quality, but also in terms of impact on the design process within the corporate environment. The second part of the chapter describes the application of Response Surface Models introduced in Chap. 4 to a tiled, multiple-instruction, many-core architecture developed by the Chinese Academy of Sciences. Overall, the results have showed that different models can present a trade-off of accuracy versus computational effort. In fact, throughout the evaluation, we observed that high accuracy models require high computational time (for both model construction time and prediction time); vice-versa low model construction and prediction time has led to low accuracy.

Chapter 9 reports a case study of DSE for supporting Run-time Resource Management (RRM). The management of system resources for an MPSoC dedicated to multiple MPEG4 encoding is addressed in the context of an Automotive Cognitive Safety System. The runtime management problem is defined as the minimization of the platform power consumption under resource and Quality of Service (QoS) constraints. The chapter provides an insight of both, design-time and run-time aspects of the problem. During the preliminary design-time DSE phase, the best configurations of run-time tunable parameters are statically identified for providing the best trade-offs in terms of run-time costs and application QoS. To speed up the optimization process without reducing the quality of final results, a multi-simulator framework is used for modeling platform performance. At run-time, the RRM exploits the

design-time DSE results for deciding an operating configuration to be loaded for each MPEG4 encoder. This operation is carried out dynamically, by following the QoS requirements of the specific use-case.

Due to the large number of topics discussed in the book and their heterogeneity, the background on system modeling, simulation and exploration is discussed chapter by chapter with a separate reference set for each chapter. This choice also contributed to make each chapter self-contained.

Overall, we believe that the book chapters cover a set of definitely important and timely issues impacting the present and future research on automatic DSE for embedded multi-core on-chip architectures. We sincerely hope that the book could become a solid reference in the next years. In our vision, the authors put a big effort in clearly presenting their technical contributions outlining the potential impact and benefits of the proposed approach on same case studies. Our warmest gratitude goes to the MULTICUBE team, for their continuous effort and dedication during the project and for their contribution as authors of the chapters. We would like to gratefully acknowledge our EC Project Officer, Panagiotis Tsarchopoulos and our EC Project Reviewers: Alain Perbost, Andrzej Pulka and Kamiar Sehat for their valuable comments and guidance during the MULTICUBE project. A special thanks to Charles Glaser from Springer for encouraging us to write a single textbook on the topic of design space exploration based on our experience of the MULTICUBE project.

With our work on MULTICUBE project and this book, we have pushed towards the adoption of automatic design space exploration for the design of multi-processor architectures for embedded computing systems. This book is expected to be one of the most important dissemination vehicles to spread out the knowledge developed in the MULTICUBE project in the international community after the end of the project.

The Editors,
Cristina Silvano
William Fornaciari
Eugenio Villar

Milano, Italy
Milano, Italy
Santander, Spain
March 2011

Contents

Part I Methodologies and Tools

1 The MULTICUBE Design Flow 3
Cristina Silvano, William Fornaciari, Gianluca Palermo, Vittorio Zaccaria, Fabrizio Castro, Marcos Martinez, Sara Bocchio, Roberto Zafalon, Prabhat Avasare, Geert Vanmeerbeeck, Chantal Ykman-Couvreur, Maryse Wouters, Carlos Kavka, Luka Onesti, Alessandro Turco, Umberto Bondi, Giovanni Mariani, Hector Posadas, Eugenio Villar, Chris Wu, Fan Dongrui, and Zhang Hao

2 M3-SCoPE: Performance Modeling of Multi-Processor Embedded Systems for Fast Design Space Exploration 19
Hector Posadas, Sara Real, and Eugenio Villar

3 Optimization Algorithms for Design Space Exploration of Embedded Systems 51
Enrico Rigoni, Carlos Kavka, Alessandro Turco, Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, and Giovanni Mariani

4 Response Surface Modeling for Design Space Exploration of Embedded System 75
Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, Enrico Rigoni, Carlos Kavka, Alessandro Turco, and Giovanni Mariani

5 Design Space Exploration Supporting Run-Time Resource Management 93
Prabhat Avasare, Chantal Ykman-Couvreur, Geert Vanmeerbeeck, Giovanni Mariani, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria

6 Run-Time Resource Management at the Operating System Level 109
Patrick Bellasi, Simone Corbetta, and William Fornaciari

Part II Application Domains

7 High-Level Modeling and Exploration of a Powerline Communication Network Based on System-on-Chip 145
Marcos Martinez, David Ferruz, Hector Posadas, and Eugenio Villar

8 Design Space Exploration of Parallel Architectures 171
Carlos Kavka, Luka Onesti, Enrico Rigoni, Alessandro Turco, Sara Bocchio, Fabrizio Castro, Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, Giovanni Mariani, Fan Dongrui, Zhang Hao, and Tang Shibin

9 Design Space Exploration for Run-Time Management 189
Giovanni Mariani, Chantal Ykman-Couvreur, Prabhat Avasare, Geert Vanmeerbeeck, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria

Conclusions 205

Index of Terms 207

List of Contributors

Prabhat Avasare IMEC, Leuven, Belgium
e-mail: avasare@imec.be

Patrick Bellasi Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: bellasi@elet.polimi.it

Sara Bocchio STMicroelectronics, Agrate, Italy
e-mail: sara.bocchio@st.com

Umberto Bondi ALaRI, University of Lugano, Lugano, Switzerland
e-mail: umberto.bondi@usi.ch

Fabrizio Castro Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: castro@elet.polimi.it

Simone Corbetta Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: scorbetta@elet.polimi.it

Fan Dongrui Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
e-mail: fandr@ict.ac.cn

David Ferruz Design of Systems on Silicon (DS2), Valencia, Spain
e-mail: david.ferruz@ds2.es

William Fornaciari Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: fornacia@elet.polimi.it

Zhang Hao Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
e-mail: zhanghao@ict.ac.cn

Carlos Kavka ESTECO, Trieste, Italy
e-mail: carlos.kavka@esteco.com

Giovanni Mariani ALaRI, University of Lugano, Switzerland
e-mail: giovanni.mariani@usi.ch

Marcos Martinez Design of Systems on Silicon (DS2), Valencia, Spain
e-mail: marcos.martinez@ds2.es

Luka Onesti ESTECO, Trieste, Italy
e-mail: luka.onesti@esteco.com

Gianluca Palermo Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: gpalermo@elet.polimi.it

Hector Posadas University of Cantabria, Santander, Spain
e-mail: posadash@teisa.unican.es

Sara Real University of Cantabria, Santander, Spain
e-mail: realsara@teisa.unican.es

Enrico Rigoni ESTECO, Trieste, Italy
e-mail: rigoni@esteco.it

Tang Shibin Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
e-mail: tangshibin@ict.ac.cn

Cristina Silvano Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: silvano@elet.polimi.it

Alessandro Turco ESTECO, Trieste, Italy
e-mail: alessandro.turco@esteco.it

Geert Vanmeerbeeck IMEC, Leuven, Belgium
e-mail: vanmeerb@imec.be

Eugenio Villar University of Cantabria, Santander, Spain
e-mail: villar@teisa.unican.es

Maryse Wouters IMEC, Leuven, Belgium
e-mail: woutersm@imec.be

Chris Wu STMicroelectronics, Beijing, China
e-mail: chris.wu@st.com

Chantal Ykman-Couvreur IMEC, Leuven, Belgium
e-mail: ykman@imec.be

Vittorio Zaccaria Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: zaccaria@elet.polimi.it

Roberto Zafalon STMicroelectronics, Agrate, Italy
e-mail: roberto.zafalon@st.com

Abbreviations

4CIF	4 × Common Intermediate Format
ACCS	Automotive Cognitive Safety System
ADRES	Architecture for Dynamically Reconfigurable Embedded System
ADRS	Average Distance from Reference Set
API	Application Programming Interface
APRS	Adaptive-windows Pareto Random Search
BP	Bitstream Packetizing
CMP	Chip Multi Processor
CSU	Central Safety Unit
DOE	Design of Experiments
DSE	Design Space Exploration
DSP	Digital Signal Processor
DVFS	Dynamic Voltage and Frequency Scaling
EC	Entropy Coding
ES	Evolution Strategies
GA	Genetic Algorithm
HW	Hardware
ILP	Integer Linear Programming
IP	Intellectual Property
IPC	Instruction per Cycle
MC	Motion Compensation
ME	Motion Estimation
MFGA	Magnifying Front GA
MMKP	Multi-dimension Multiple-choice Knapsack Problem
MOGA	Multi Objective GA
MOO	Multi-Objective Optimization
MOPSO	Multi Objective Particle Swarm Optimization
MOSA	Multi Objective Simulated Annealing
MPA	MPSoC Parallelization Assist
MPEG4	Moving Picture Experts Group 4
MPSoC	Multi-Processor Systems on Chip
NSGA-II	Non-dominated Sorting Genetic Algorithm, second version

OS	Operating System
QoS	Quality of Service
RM	Resource Manager
RRM	Run-time Resource Management
RSM	Response Surface Model
RTOS	Run-Time Operating System
RTRM	Run-Time Resource Manager
SoC	System on Chip
STM	STMicroelectronics
SW	Software
TC	Texture Coding
TCM	Task Concurrency Management
TLM	Transaction-Level Model
TU	Texture Update
VLIW	Very Long Instruction Word
XML	eXtensible Markup Language

About the Editors

Cristina Silvano received the M.S. degree in Electronic Engineering from Politecnico di Milano, Milano, Italy, in 1987 and the Ph.D. degree in Computer Engineering from the University of Brescia, Brescia, Italy, in 1999. From 1987 to 1996, she was a Senior Design Engineer at R&D Labs, Groupe Bull, Pregnana, Italy. From 2000 to 2002, she was Assistant Professor at the Department of Computer Science, University of Milan, Milano. She is currently Associate Professor (with tenure) in Computer Engineering at the Dipartimento di Elettronica e Informazione, Politecnico di Milano. She has published two scientific international books and more than 90 papers in international journals and conference proceedings, and she is the holder of several international patents. Her primary research interests are in the area of computer architectures and computer-aided design of digital systems, with particular emphasis on design space exploration and low-power design techniques for multiprocessor systems-on-chip. She participated to several national and international research projects, some of them in collaboration with STMicrolectronics. She is currently the European Coordinator of the project FP7-2PARMA-248716 on "PARallel PAradigms and Run-time MAnagement techniques for Many-core Architectures" (Jan. 2010–Dec. 2012). She was also the European Coordinator of the FP7-MULTICUBE-216693 project on "Multi-objective design space exploration of multi-processor SoC architectures for embedded multimedia applications" (Jan. 2008–June 2010). She served as member and/or co-chair in the technical committees of several international conferences such as MICRO, DAC, DATE, NOCS, SASP, ARCS and VLSI-SOC.

William Fornaciari is Associate Professor (with tenure) at Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy. He received the Laurea (M.Sc.) degree in Electronic Engineering and the Ph.D. degree in Automation Engineering and Computer Sciences, both from the Politecnico di Milano in 1989 and 1993 respectively. He has published five books and over 100 papers in international journals and conference proceedings, collecting four best paper awards, one certification of appreciation from IEEE and holds two international patents. Since 1993 he is member of program and scientific committees and chair of international conferences in the field of computer architectures, EDA and system-level design. He has been also

involved in the faculty of a joint Master program between the Politecnico di Milano and the University of Chicago at Illinois. Since 1997 has been involved in 11 EU-funded international projects and has been part of the pool of experts of the Call For Tender No. 964-2005 - WING - Watching IST INnovation and knowledGe. His current research interest includes embedded systems design methodologies, real-time operating systems, energy-aware design of SW and HW for multi-many core systems, reconfigurable computing and wireless sensor networks. Recently his involvement is manily releated to MULTICUBE, SMECY, 2PARMA and COMPLEX european projects.

Eugenio Villar got his Ph.D. in Electronics from the University of Cantabria in 1984. Since 1992 is Full Professor at the Electronics Technology, Automatics and Systems Engineering Department of the University of Cantabria where he is currently the responsible for the area of HW/SW Embedded Systems Design at the Microelectronics Engineering Group. His research activity has been always related with system specification and modeling. His current research interests cover system specification and design, MPSoC modeling and performance estimation using SystemC and UML/Marte. He is author of more than 100 papers in international conferences, journals and books in the area of specification and design of electronic systems. Prof. Villar served in several technical committees of international conferences like the VHDL Forum, Euro-VHDL, EuroDAC, DATE, and FDL. He has participated in several international projects in electronic system design under the FP5, FP6 and FP7, Itea, Medea and Artemis programs. He is the representative of the University of Cantabria in the ArtemisIA JU.

Part I
Methodologies and Tools

Chapter 1
The MULTICUBE Design Flow

Cristina Silvano,William Fornaciari, Gianluca Palermo, Vittorio Zaccaria, Fabrizio Castro, Marcos Martinez, Sara Bocchio, Roberto Zafalon, Prabhat Avasare, Geert Vanmeerbeeck, Chantal Ykman-Couvreur, Maryse Wouters, Carlos Kavka, Luka Onesti, Alessandro Turco, Umberto Bondi, Giovanni Mariani, Hector Posadas, Eugenio Villar, Chris Wu, Fan Dongrui, and Zhang Hao

Abstract This chapter introduces the design-flow of the MULTICUBE project whose main goal is the definition of an automatic multi-objective Design Space Exploration (DSE) framework to be used to tune the parameters of System-on-Chip architectures by taking into account the target set of metrics (e.g. energy, latency, throughput, etc.). One of the important goals of the automatic multi-objective DSE framework is to find design trade-offs that best meet system constraints and cost criteria which are indeed strongly dependent on the target application. A set of heuristic optimisation algorithms have been defined to reduce the overall optimization time by identifying an approximated Pareto set of parameter configurations with respect to a set of selected figures of merit. Once the approximated Pareto set is built, the designer can quickly apply decision criteria to select the best configuration satisfying the constraints. The DSE flow is based on the interaction of two frameworks to be used at design time: the Design Space Exploration Framework, a set of open-source and proprietary architectural exploration tools, and the Power/Performance Estimation Framework, a set of modeling and simulation tools (open-source and proprietary) operating at several levels of abstraction. The DSE flow also includes the specification of an XML integration interface to connect the exploration and estimation frameworks and a Run-time Resource Manager exploiting, at run-time, the best software configuration alternatives derived at design-time to optimize the usage of system resources.

1.1 Introduction

Many point tools exist to optimise particular aspects of embedded systems. However, an overall design space exploration framework is needed to combine all the decisions into a global search space, and a common interface to the optimisation and evaluation

C. Silvano (✉)
Dipartimento di Elettronica e Informazione Politecnico di Milano, Milano, Italy
e-mail: silvano@elet.polimi.it

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_1, © Springer Science+Business Media, LLC 2011

tools. The MULTICUBE project focused on the definition of an automatic multi-objective Design Space Exploration (DSE) framework to be used to tune the System-on-Chip architecture for the target application evaluating a set of metrics (e.g. energy, latency, throughput, bandwidth, QoS, etc.) for the next generation of embedded multimedia platforms.

On one side, the MULTICUBE project defined an automatic multi-objective DSE framework to find design trade-offs that best meet system constraints and cost criteria, strongly dependent on the target application, but also to restrict the search space to crucial parameters to enable an efficient exploration. In the developed DSE framework, a set of heuristic optimisation algorithms have been defined to reduce the overall exploration time by computing an approximated Pareto set of configurations with respect to the selected figures of merit. Once the approximated Pareto set has been built, the designer can quickly select the best system configuration satisfying the target constraints.

On the other side, the MULTICUBE project defined a run-time DSE framework based on the results of the design-time exploration to optimise at run-time the allocation and scheduling of different application tasks. The design-time exploration flow results in a Pareto-optimal set of design trade-offs with different speed, energy, memory and communication bandwidth parameters. This information is used at run-time by a small OS kernel to make an informed decision about how the resources should be distributed over different tasks running on the multi-processor system on-chip. This resource distribution cannot be done during the design-time exploration itself, since it depends on which tasks are active at that time.

The goal of MULTICUBE design flow is to cover the gap between the system-level specification and the definition of the optimal application-specific architecture. The MULTICUBE activities have been driven by targeting the construction of a set of tools and accurate methodologies to support the design of application specific multi-core architectures.

In this context, a SystemC-based multi-level modeling methodology for multi-processors has been developed in the project. Once received the target architecture as input, the system model is provided to the simulator to evaluate different architectural trade-offs in terms of metrics. Then, the Design Space Exploration framework can be used to evaluate candidate configurations based on several heuristic optimisation algorithms. This step is implemented as an optimisation loop, where the selected architecture instance generated by the DSE framework is given back to the estimation framework for the metrics evaluation. The tool integration phase in MULTICUBE enabled to implement an automatic system optimisation engine to generate, for the target MPSoC architecture, either the best architectural alternative (if the exploration is done at design-time) or the best tasks scheduling and allocation (if the exploration is done at run-time).

To enable a widespread dissemination and use of the design flow in several application contexts, the following pre-requirements are introduced. First, the design flow aims at being independent from the particular language used for the description of the use case simulator. The design flow and the associated design tools should free the simulator provider from being tied to a specific programming language or

model. Second, the interface between the design tools and the use case simulators should be specified and implemented by using a widely accepted and standardized interface. Standard interfaces are characterized by being supported by a large number of parsing and validation tools either in the public domain or commercially available while enabling a faster adoption of the design tool itself. Among the available interface specification languages, the most widely accepted and flexible is XML. XML enables to create efficient, customized data structures which are, at the same time, supported by industrial and academic tools for parsing, semantic evaluation and validation. These data structures can be used to facilitate the definition of tool interfaces.

The Chapter is organized as follows. Section 1.2 provides an overview of the MULTICUBE design flow, while Sect. 1.3 describes the design tools integration based on a common interface. Finally, Sect. 1.4 presents the advantages in using the automatic design space exploration approach.

1.2 Overview of the Design Flow

The MULTICUBE DSE flow (see Fig. 1.1) is based on the interaction of two frameworks to be used at design time: the Design Space Exploration Framework, an architecture exploration set of tools, and the Power/Performance Estimation Framework, a set of modeling and simulation tools operating at several levels of abstraction. The DSE flow also includes a Run-time Resource Manager to select at run-time the best design alternatives in terms of power/performance trade-offs generated during the design-time exploration phase.

According to the exploitation plan of the MULTICUBE project, both open-source and the proprietary exploitation models and tools co-exist into a single coherent view. This has been possible by making the design tools to adopt the same common MULTICUBE XML-based interface described in Sect. 1.3.

1.2.1 The Design Space Exploration Framework

The MULTICUBE Design Space Exploration Framework (see Fig. 1.1) consists of an architecture exploration set of tools providing the designers with the most appropriate optimisation of the multi-processor SoC considering several figures of merit such as performance and energy consumption.

The Design Space Exploration tools can be used at design time to automatically identify the Pareto optimal solutions of a multi-objective exploration given a set of design space parameters. During the MULTICUBE project, two design space exploration tools and some optimisation and analytical techniques have been developed and validated.

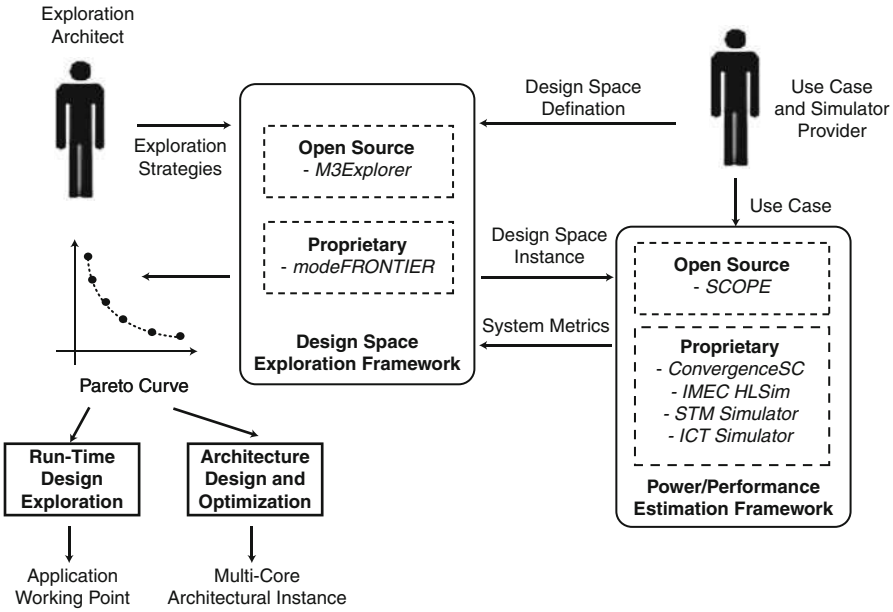

Fig. 1.1 Overview of the MULTICUBE design flow

- A new open source tool (Multicube Explorer available at http://www.multicube.eu) suitable for automatic design space exploration of MPSoC architectures. The tool enables a fast optimisation of parameterized system architecture towards a set of objective functions (e.g., energy, delay and area), by interacting with a system-level simulator through an open XML-based interface. Multicube Explorer provides a set of innovative sampling and optimisation techniques to support the designer in finding the best objective functions trade-offs. It also provides a set of Response Modeling Methods for speeding up the optimisation phase.
- An existing commercial tool (modeFRONTIER from ESTECO), widely adopted in other optimisation fields, has been re-targeted to support automatic DSE in the embedded systems field. The tool includes the most recent optimisation techniques available in literature, ranging from Design of Experiments to direct optimisers. modeFRONTIER (see also http://www.esteco.com) also provides meta-modeling support for the creation of interpolating surfaces from well statistically distributed designs to be used to perform the optimisation without computing any further analysis. The tool also supports multivariate statistical analysis and data mining tools directly integrated in the exploration process to enable the user to easily analyse complex data. The graphical user interface of modeFRONTIER provides access to all features of design experiment definition, exploration and analysis in a simple and intuitive way.

Both tools leverage a set of multi-objective optimisation algorithms that have been validated on several industrial use cases. In multi-objective optimisation problems

there are more than one objective to be optimised (maximized or minimized), meaning that the outcome of the optimisation process is not a single solution but a set of solutions. This set of solutions, which is called the Pareto front, represents the best trade-off between the different (and possibly contradictory) objectives. The set of algorithms implemented includes state-of-the-art algorithms widely used in the field of multi-objective optimisation (ranging from evolutionary and genetic algorithms up to simulated annealing and particle swarm algorithms), enhanced versions of algorithms that were specifically targeted in the project for the multi-core SoC design, and new developed algorithms. The multi-objective optimisation algorithms developed in the MULTICUBE project are described in more detail in Chap. 3 of this book.

1.2.2 The Power/Performance Estimation Framework

The Power/Performance Estimation Framework (see Fig. 1.1) consists of a methodology and related tools that have been set up to provide accurate estimates for complexity, timing and power consumption at different abstraction levels and for different use cases. A set of tools has been used for the system modeling and estimation of several metrics such as energy consumption and execution time of the target MPSoC platforms among which:

- Multicube SCoPE: an extension of the open-source high-level SCoPE performance and power evaluation framework [7] developed by University of Cantabria for performing HW/SW co-simulation. Multicube SCoPE enables the definition of SystemC platform template models to evaluate performance and power consumption. Multicube SCoPE efficiency comes from the fact that performance and power estimations of the software side are performed at the application source code level by using back-annotation. The back-annotated software components are then linked to the hardware components by using standard SystemC interfaces. This modeling style is called *Timing Approximate*. Software back-annotation avoids instruction set simulation therefore decreasing of several orders of magnitude the simulation time and maintaining a fairly good accuracy with respect to cycle-accurate simulation. Multicube SCoPE also provides some hooks for enabling C/C++ software code to invoke operating system primitives compliant with POSIX and MicroC/OS. Multicube SCoPE is described in more detail in Chap. 2 of this book.
- A proprietary set of simulation tools developed by IMEC as SystemC-based transaction-level multi-core simulator built on top of the CoWare virtual prototyping environment to support platform-based design approach. The TLM-based prototype models an ADRES multi-core [6] and has been integrated with both modeFRONTIER and Multicube Explorer tools. The platform is composed of a variable number of processor nodes and memory nodes. All processor nodes

contain the IMEC proprietary ADRES VLIW processor and its scratch-pad local data (L1) memory. The processing nodes are connected to the memory nodes by a configurable communication infrastructure. It can be either a multi-layer AHB bus, which provides a full point-to-point connectivity, or a NoC model built with the CoWare AVF cross-connect IP.

- A High-Level time-annotated Simulator (HLSim, developed by IMEC) that provides a fast simulator of the ADRES platform at higher abstraction level to estimate metrics like performance and power consumption for a given platform architecture executing a parallelized version of the application. During the MULTICUBE project, HLSim has been extended with metrics on energy consumption derived from a multimedia use case for a relative comparison between different architectures and parallelizations. The introduction of HLSim in the design flow has provided several benefits such as speeding up the simulation and starting up the design exploration earlier than planned. HLSim-based explorations are much faster than TLM-based ones so as more extensive DSE was done by using HLSim to extract Pareto set information to be used at run-time.
- An instruction set simulator has been used for SP2 superscalar processor provided by STMicroelectronics and one simulator for the many-core architecture provided by ICT Chinese Academy of Science. Both simulators expose program execution time and power consumption as system-level metrics. More in detail, the ICT many-core architecture is a tiled MIMD machine composed of a bi-dimensional grid of homogeneous, general-purpose compute elements, called *tiles*. A 2D-mesh network architecture is used for connecting the cores to a non-shared memory sub-system.
- The DS2's STORM platform, a control-oriented architecture for powerline communication. The platform is used to model a PLC (Programmable Logic Controller) technology with several implementation choices. For this platform, both Ethernet QoS and internal communication are considered as metrics.

Given these target simulators, the MULTICUBE project developed a methodology, the multi-simulator based DSE approach shown in Fig. 1.2, to avoid potentially sub-optimal DSE results and to speed up the DSE process by exploiting multiple platform simulators to run the application at different abstraction levels.

The main idea is to get timing information (in terms of processor cycles) for an application execution on an accurate simulator (e.g. TLM-based cycle-accurate simulator) and feed this timing information back to a high-level timed simulator (e.g. HLSim) to achieve validation across simulators. Then, the DSE is done with a large number of application runs by using faster higher-level simulators (e.g. HLSim) and then the derived interesting operating points (usually clusters of operating points) are refined by using more accurate simulators (e.g. TLM-based and/or cycle-accurate simulators). The proposed methodology exploiting the synergy of multiple simulators at several abstraction level can be used to further speed up the DSE process while guaranteeing good accuracy of the simulation results. The methodology has been validated for the MPEG4 encoder application provided by IMEC by using three different

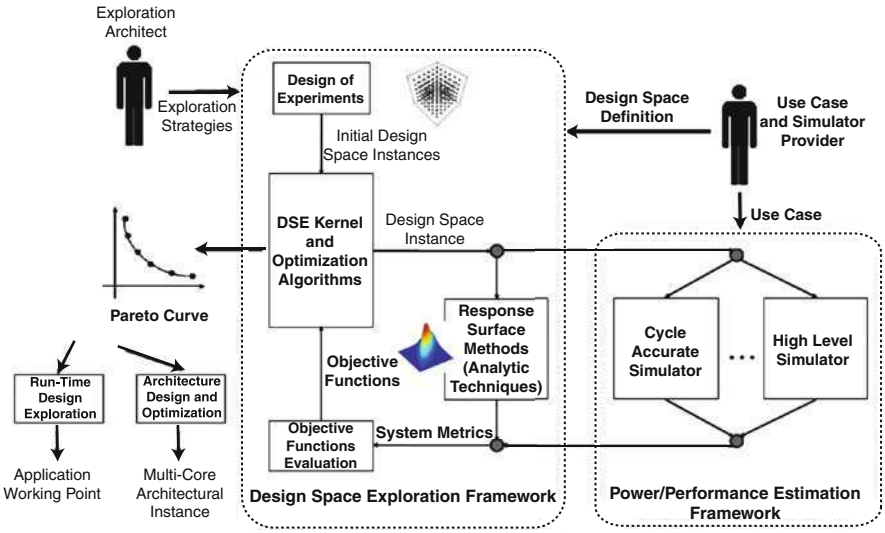

Fig. 1.2 Overview of the multi-simulator based DSE design flow

simulators (Multicube SCoPE, HLSim and TLM-based simulator) interfaced with the two available DSE tools (modeFRONTIER and Multicube Explorer). Overall, an acceptable relative accuracy has been found [1] with a significant speed-up in simulation time.

1.2.3 Response Surface Modeling Techniques

A set of analytical techniques (Response Surface Models, or RSMs) have been introduced to further speed up the design space exploration process (see Fig. 1.2). These techniques are key factors for developing a model of the system behavior without requiring the simulation of all the possible design configurations. RSMs have been proved to be an effective solution for analytically predicting the behavior of the system in terms of the target metrics without resorting to the system simulation.

Response Surface Modeling (RSM) techniques leverage the analytical dependence between several design parameters and one or more response variables by adopting both interpolation and regression techniques. The basic principle is to use a set of simulations either generated ad hoc by a Design of Experiment (DoE) phase or obtained by an exploration strategy previously applied to the design space, in order to obtain a response model of the system behavior. In the project, several RSM techniques have been implemented, among them Radial Basis Functions [3], Linear Regression [4, 5], Artificial Neural Networks [2] and Shepard's Interpolation. Every

RSM presented dramatic speed-up in terms of evaluation. Besides, it has been found that a peculiar mathematical transformation of input training set known as Box-Cox λ transform [4] has a great impact on the prediction accuracy. A sub-set of the above analytical techniques has been implemented and integrated in the MULTICUBE open-source tool while another sub-set was already available in the modeFRONTIER tool. RSM techniques for DSE are described in more detail in Chap. 4 of this book.

1.2.4 Run-Time Resource Management

The MULTICUBE design flow has been built to provide not only design-time support but also run-time support. In this scenario, at design time, the multi-objective design space exploration framework generates a set of Pareto-optimal operating points (for each application) annotated with system metrics like energy consumption, execution time, and memory and communication bandwidth values. The Pareto set can then be exploited at run time (while the application(s) are running) to optimise the overall system behavior (see Fig. 1.3). Specifically a separate Run-time Resource Manager (RRM) has been developed to exploit the set of operating points derived at design-time for all applications to steer the overall system behavior according to the imposed user requirements (quality, power, performance, etc.). The goal of the RRM is to use the Pareto information given by the design-time exploration on the operating points (of all applications) to make at run-time a decision to allocate the system resources to active applications based on the user requirements in terms of Quality of Service. Run-time Resource Management techniques are described in Chap. 5, while some more general concepts about resources management at the Operating System layer are presented in Chap. 6.

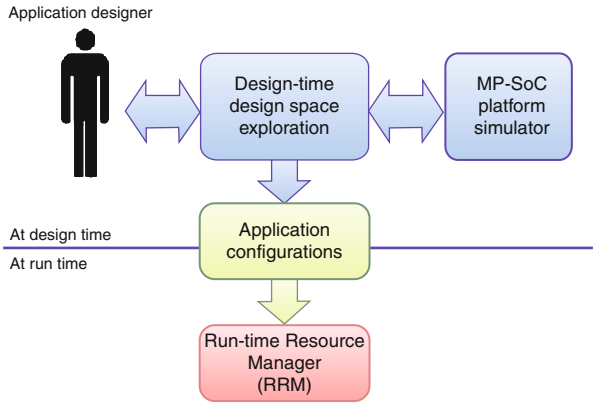

Fig. 1.3 Overview of the run-time support

1.3 Design Tool Integration based on the MULTICUBE XML Interface

Strategic importance from the point of view of the MULTICUBE exploitation is associated to the common XML Tool Interface Specification for the integration of the different tools and use cases. The common interface enabled the independent development of software modules and a seamless integration of design tools and data structures into a common design environment. The specification is defined in terms of XML, a widely used standard notation. To introduce the notation, let us highlight that, in the MULTICUBE design flow, there are two types of user agents to interact with the framework: the *use case and simulator provider* and the *exploration architect*. The simulator is the executable model of the use case and it is a single executable file (binary or script) which interacts with the DSE tool to provide the value of the estimated metrics, given an input configuration. The MULTICUBE project addressed the formalization of the interaction between the simulator and the DSE tools, that is essentially an automatic program-to-program interaction (see Fig. 1.4):

1. The DSE tool generates one feasible system configuration whose system metrics should be estimated by the simulator.
2. The simulator generates a set of system metrics to be fed back to the DSE tool.

To automatically link the use case simulator to the DSE tool, a design space definition file should be released by the use case and simulator provider together with the executable model of the use case (simulator). This file describes the set of configurable parameters of the simulator, their values range and the set of evaluation metrics that can be estimated by the simulator. This file describes also how to invoke the simulator as well as an optional set of rules with which the generated parameter values should be compliant. The rules are only used by the exploration tool to avoid the generation of invalid or unfeasible solutions during the automated exploration

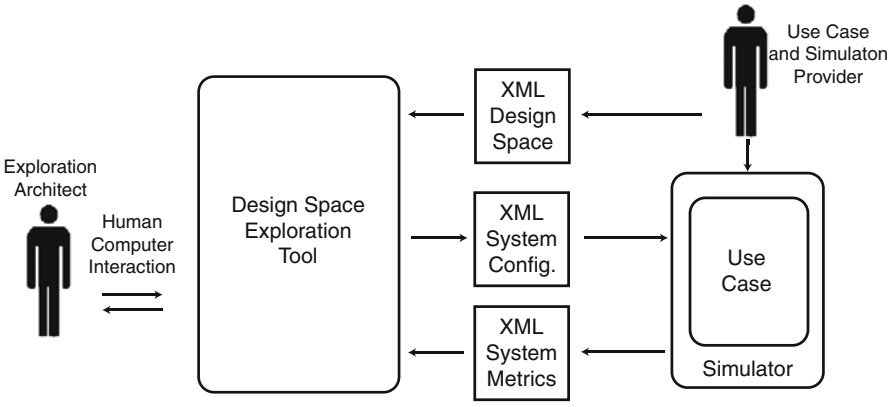

Fig. 1.4 Overview of the tool interfaces via XML

process. The above interaction has been addressed by creating a specification based on an XML based grammar for writing both the design space definition file and the simulator interface files. The grammar is defined by using the XSD schema language.

1.3.1 Design Space Definition

The definition of the design space is done by using an XML file that is composed of a preamble, which defines the name-space and supported version. The current release of the MULTICUBE XML interface specification is R1.4 and it is available on MULTICUBE web page (www.multicube.eu).

```
<?xml version="1.0" encoding="UTF-8"?>
<design_space xmlns="http://www.multicube.eu/" version="1.4">
<simulator> ... </simulator>
<parameters> ... </parameters>
<system_metrics> ... </system_metrics>
<rules> ... </rules>
</design_space>
```

The remaining part of the file describes the simulator invocation method (<simulator> tag), the set of parameters of the simulator which can be configured (<parameters> tag), the system metrics which can be estimated by the simulator (<system_metrics> tag) and the rules which have to be taken into account by the exploration engine to generate the feasible configurations.

1.3.1.1 Simulator Invocation

The <simulator_executable> marker is used for specifying the complete path name of the executable:

```
<simulator>
<simulator_executable path="/path/my_simulator_executable" />
</simulator>
```

1.3.1.2 Parameters Definition

The <parameters> tag is used by the use case and simulator provider to specify the names, the types and the ranges of the parameters that can be explored by the DSE tool. The section contains a list of <parameter> markers:

```
<parameter>
<parameter name="il1_cache_block_size_bytes"
   description="..." type="exp2" min="8" max="64"/>
<parameter name="bpred" description="b.p. type" type="string">
   <item value="nottaken"/>
   <item value="taken"/>
   <item value="perfect"/>
   <item value="bimod"/>
```

```
9       <item value="2lev"/>
10      <item value="comb"/>
11  </parameter>
12  ...
13  </parameters>
```

For each parameter an unique name must be provided. The parameters types can be divided into two categories: scalar types, variable vector types. Scalar types can be **integer**, **boolean** (a subset of integers), **exp2** (progression of power of 2) and **string** (a type for defining categorical variables). Vector types can be used to describe combination of boolean values (*on-off-masks* or *permutations*). In particular, on-off-masks can be useful for describing the space of active processors while permutations can be used to describe the mapping of tasks on the available processors.

1.3.1.3 System Metrics Definition

The <system_metrics> section is used by the use case and simulator provider to specify the names, the types and the units of the system metrics that can be estimated by the simulator:

```
1  <system_metrics>
2  <system_metric name="cycles" type="integer" unit="cycles" />
3  <system_metric name="instructions" type="integer" unit="insts"/>
4  <system_metric name="powerconsumption" type="float" unit="W" />
5  <system_metric name="area" type="float" unit="mm2" />
6  </system_metrics>
```

A complex expression of the system metrics can be defined as one of the objective of the exploration algorithm.

1.3.2 Simulator Input/Output XML Interface

The simulator input file contains a preamble and a sequence of <parameter> sections where, for each parameter, the name and the value is specified. The number of <parameter> sections and the name of the parameters should be the same as defined in the XML Design Space description file. Similarly the simulator output file contains a preamble and a sequence of <system_metric> sections where, for each metric, the name and the value is specified. Besides, an appropriate error reporting syntax has been described in the specification.

1.4 Advantages of Automatic DSE

The procedure to assess the benefits of the introduction of an Automatic Design Space Exploration (or Optimization Methodology) has to address, not only the final objective quality and the improvement of the target design but also the impact of such a technology on the entire design process of embedded computing platforms. The

benefits on the process can be measurable and tangible like the reduction of the overall design process lead time, and qualitative or intangible like the streamlining and the reduction of human error prone repetitive operations. These benefits are particularly valuable for design problems where the number of configuration parameters to be explored is quite large, like in MP-SoC designs.

The specific design process activities can be analyzed and classified to measure the various performance indicators. In a general way, we can consider the following steps as the basis for any manual design space exploration or optimization process:

- Model Setup: preparation of an initial model of the virtual platform;
- Simulation: execution of the simulation of the executable model with a single configuration of parameters;
- Results Assessment: meaningful measures are extracted and compared with historical and expected ones;
- Model Edit: the model is manually modified and resubmitted for a further analysis.

Figure 1.5 represents the steps of a typical manual exploration procedure. In a manual approach, the exploration of the design space is done by subjective assumptions of the human designer, who will modify at most one or two parameters per evaluation. The model simulation corresponds to a minimal portion of the time of the whole exploration procedure. A large amount of time is spent by the designer editing the configuration parameters and analyzing the results. There is also an Idle Time (from the point of view of the use of computational resources) that lasts from the end of the simulation till the moment in which the human operator is informed about it and handles the simulation tools to get the results. This idle time can be very short if the designer is immediately informed about the end of the simulation or can be significant if the designer is not on duty.

The automatic design space exploration process can be defined by identifying the following basic steps:

- Model setup: the model must be correctly parameterized in order to be easily managed by the automatic exploration tools;

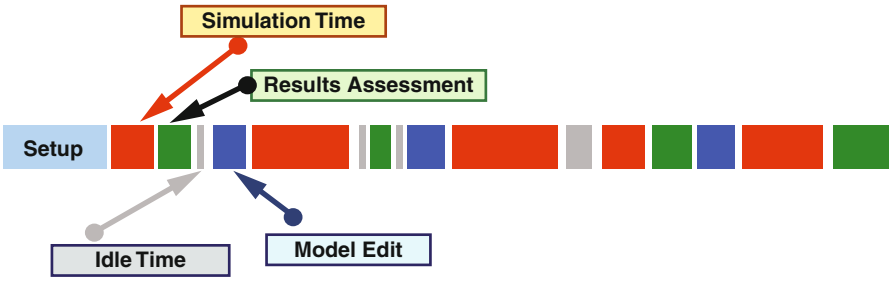

Fig. 1.5 Manual exploration procedure

- Simulation time: the simulation of the executable model with a single configuration of parameters is carried out (this phase is similar to the one that is performed with the manual approach);
- Automatic DSE overhead: this step includes the automatic assessment of the results, the automatic selection of the next configuration to be simulated (model selection) and the data transfer operations between the simulator and the design space exploration tool and its corresponding storage into the design database.

Figure 1.6 describes the automatic exploration procedure. The exploration of the design space is done by numerical/objective criteria. The Design Space Exploration tool (or "exploration engine") will change systematically all the parameters for each analysis and will evaluate the best result by adopting numerical formulas. The setup phase can be considerably longer than the set up of a manual exploration, since it usually requires an extended definition of the model to interact with the exploration engine, the definition of the proper optimization strategy and the definition of the multi-objective goals to be achieved. However, after that, the model evaluations (Simulation Time) is similar to the simulation time required in the manual approach, except for a further step (Automatic DSE overhead) that involves the automatic assessment of the results, the automatic selection of the next configuration to be simulated (model selection) and the time spent for data communication and storage.

Based on the past experience of ESTECO to deal with industrial customers, even if a single evaluation takes just a few seconds, it is difficult that a designer using a manual optimization procedure can evaluate more than seven designs of medium complexity within one hour. The designer has to go through all steps described before, editing the configuration parameters, running the analysis, reading and analyzing the results, etc. In this case, it is expected that in one person-day (10 h), the designer can run at most 70 designs. For the same problem, experience shows that an automatic approach can handle something like 600 designs per hour, which means about 14,400 designs per day. Since the automatic procedure can work 24 four hours a day, including weekends and holidays, the advantages of the automatic procedure are very clear.

However, there are other advantages of the automatic exploration procedure. In the automatic exploration, all data concerning previous evaluations are always stored in a

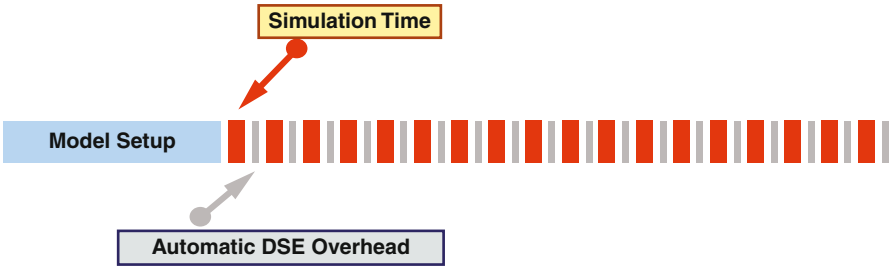

Fig. 1.6 Automatic exploration procedure

structured database. The designer, not only will not be stuck on repetitive operations, but can focus his/her attention (and profit from his/her experience) in analyzing the designs database in a statistical manner.

Moreover, the automatic exploration is driven by an optimization engine based on several optimization algorithms, whereas the manual exploration is based on designer ability and experience to assess the results and to move towards the next instance of the model to be simulated.

Figure 1.7 presents a direct comparison between manual and automatic optimization. The global lead time (Tmo) of the manual design exploration is determined by the number of manual iterations that are needed to reach the expected design improvement. The global lead time (Tao) in the automatic procedure case is dominated by the effective simulation time needed by the optimization strategy, given the overhead due to the automatic extraction and processing of measures, and communication latencies between the design exploration tool and the simulator. Another difference concerns the type of results produced by both types of optimization: the automatic optimization generates a set of designs that are likely to belong to the Pareto front, while the manual optimization generates just a set of designs that the designer found interesting. On the average, the manual approach suffers from the fact that it is affected by personal views, experience and background. This decreases the likelihood of finding points that belong to the Pareto front.

The reduction of the lead time of the design exploration phase is frequently reflected in the reduction of the overall time-to-market for the final product. The lead time must be always considered as one of the objectives for the introduction of the automatic exploration methodologies in industrial design processes, together with the more specific product related ones.

The general automatic DSE procedure described in this Chapter represented the common basis for the validation of each of the use cases of the MULTICUBE project. In order to apply this procedure, the responsible for each of the use cases adapted

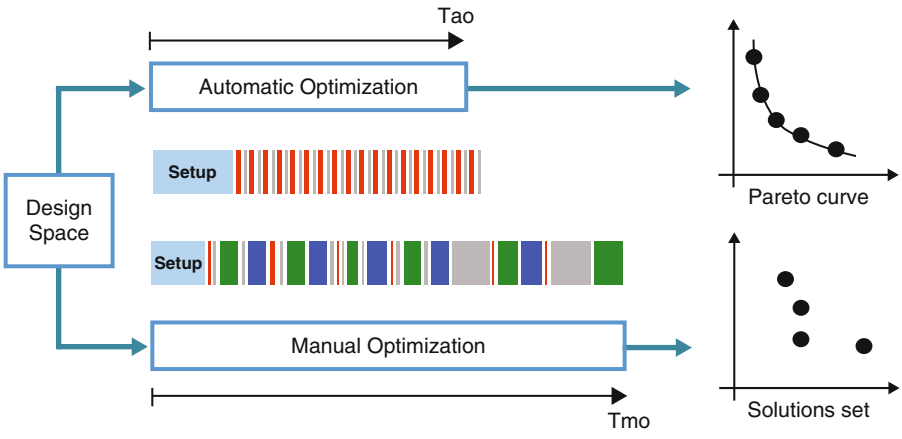

Fig. 1.7 Comparison between manual and automatic exploration procedures

however the general procedure to the specificities of its particular application and context, but without modifying the aim and the general steps described above.

1.5 Conclusions

In this chapter, the main structure of the MULTICUBE design flow has been introduced to be detailed in the next chapters. The founding principles of the proposed structure are meant to cover the gap between the system-level specification and the definition of the optimal application-specific architecture. The flow is based on the interaction of two frameworks to be used at design time: the Design Space Exploration Framework, an architecture exploration set of tools, and the Power/Performance Estimation Framework, a set of modeling and simulation tools operating at several levels of abstraction. The DSE flow also includes a Run-time Resource Manager able to select at run-time the best design alternatives in terms of power/performance trade-offs generated during the design-time exploration phase.

References

1. Avasare, P., Vanmeerbeeck, G., Kavka, C., Mariani, G.: Practical approach to design space explorations using simulators at multiple abstraction levels. In: Design Automation Conference (DAC) User Track Sessions. Anaheim, USA (2010)
2. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press (2002)
3. Joseph, P., Vaswani, K., Thazhuthaveetil, M.: Construction and use of linear regression models for processor performance analysis. High-Performance Computer Architecture, 2006. The Twelfth International Symposium on pp. 99–108 (2006)
4. Joseph, P.J., Vaswani, K., Thazhuthaveetil, M.J.: A predictive performance model for superscalar processors. In: MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 161–170. IEEE Computer Society, Washington, DC, USA (2006). DOI http://dx.doi.org/10.1109/MICRO.2006.6
5. Lee, B.C., Brooks, D.M.: Accurate and efficient regression modeling for microarchitectural performance and power prediction. Proceedings of the 12th international conference on Architectural support for programming languages and operating systems **40**(5), 185–194 (2006). DOI http://doi.acm.org/10.1145/1168917.1168881
6. Mei, B., Sutter, B., Aa, T., Wouters, M., Kanstein, A., Dupont, S.: Implementation of a coarse-grained reconfigurable media processor for avc decoder. J. Signal Process. Syst. **51**(3), 225–243 (2008). DOI http://dx.doi.org/10.1007/s11265-007-0152-8
7. Posadas, H., Castillo, J., Quijano, D., Fernandez, V., Villar, E., Martinez, M.: SystemC platform modeling for behavioral simulation and performance estimation of embedded systems. Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation pp. 219–243 (2010)

Chapter 2
M3-SCoPE: Performance Modeling of Multi-Processor Embedded Systems for Fast Design Space Exploration

Hector Posadas, Sara Real, and Eugenio Villar

Abstract Design Space Exploration for complex, multi-processor embedded systems demands new modeling, simulation, performance estimation tools and design methodologies. Recently approved as IEEE 1666 standard, SystemC has proven to be a powerful language for system modeling and simulation. In this chapter, M3-SCoPE, a SystemC framework for platform modeling, SW source-code behavioral simulation and performance estimation of multi-processor embedded systems is presented. Using M3-SCoPE, the application SW running on the different processors of the platform can be simulated efficiently in close interaction with the rest of the platform components. In this way, fast and sufficiently accurate performance metrics of the system are obtained. These metrics are then delivered to the DSE tools to evaluate the quality of the different configurations in order to select the best ones.

2.1 Introduction

System exploration with an optimum trade-off between performance and cost requires analyzing the performance of a large number of system configurations with a wide set of parameters, such as number and type of processors, memory architecture and sizing, mapping of SW tasks and suitability of communication infrastructure.

System simulation is a key design task widely used for design verification and evaluation. The main role of system simulation in embedded system design is to ensure the functional correctness of the design at the different abstraction levels. In Design Space Exploration (DSE), system simulation is used for performance analysis, providing to the exploration tool the required metrics such as delays, throughput, utilization rates, bandwidths, etc. Power consumption is becoming an additional, increasingly important metric to be estimated.

H. Posadas (✉)
University of Cantabria, Santander, Spain
e-mail: posadash@teisa.unican.es

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_2, © Springer Science+Business Media, LLC 2011

Design Space Exploration requires auxiliary tools to provide the exploration engines with the metrics needed to evaluate the different system configurations. For evaluating complex HW/SW MPSoC systems [30], very flexible evaluation mechanisms are required. Static mechanisms are adequate to evaluate the effect of different parameter values in well known architectures. The analysis of internal processor components or cache configurations are examples in that context. However, to evaluate unknown or very flexible architectures, analysis methods based on mathematical equations are not applicable. Evaluation mechanisms based on simulation are then selected.

Simulation environments for DSE have to overcome several challenges. Mainly, these simulations require very fast speeds, considering the large amount of points to be simulated. Thus, modeling techniques need to evaluate all the configurations selected by the DSE tools without provoking additional delays. While HW simulation can be performed at different abstraction levels using appropriate languages such as VHDL, Verilog, System-Verilog and SystemC [58], efficient and sufficiently accurate SW simulation requires additional efforts. Electronic System Level (ESL) [4] has been proposed as an adequate abstraction level for complete system simulations [38]. At this level, there are three main methodologies used for SW simulation: Instruction-Set Simulation (ISS), virtualization with binary translation and native co-simulation.

The first solution, ISS-based HW/SW co-simulation, is the main industrial platform simulation technology supported by mature commercial tools offered by all the major vendors [40, 56]. Currently available commercial modeling and simulation tools are based on previous research activity in academia. In [7] a generic design environment for multiprocessor system modeling was proposed. The environment enables transparent integration of ISSs and prototyping boards. As an evolution of this work, in [8] a SystemC infrastructure was developed to model architectures with multiple ARM cores. This approach provides a set of tools that enables designers to efficiently design SW applications (OS ports, compilers, etc.). A fully operational Linux version for embedded systems was ported on the platform. Software simulation was based on an ISS for each processor, thus presenting the advantages and disadvantages commented above. Moreover, it cannot be easily used to evaluate platforms that do not contain ARM processors.

Several approaches have been proposed to improve the state-of-the-art of commercial tools (Fig. 2.1). One of them is the modification of the OS running over the ISS. As the OS is in fact the interface between SW applications and the rest of the system, it can be used to save simulation time. In [62], a technique based on virtual synchronization is presented to speed up execution of several SW tasks in the ISS. Only the application tasks run over the ISS. The OS is modeled in the co-simulation back-plane thus accelerating its simulation. As the OS execution time is only part of the total execution time, the gain is limited.

A recent technology proposed for SW simulation is using virtualization with binary translation. The most representative virtualization technology is QEMU [54]. SW emulation is based on virtualization. The binary code of the target processor is dynamically translated to the host executing the same functionality. In its original

Simulation type	Speed	Accuracy
Functional execution	100000	
Timed native co-simulation	10000	
Timed binary translation	1000	
ISS (instructions)	100	
ISS (cycle accurate)	10	
Pin accurate	1	

Fig. 2.1 Approximate comparison of orders of magnitude for the different simulation mechanisms

version, QEMU does not support execution time estimation for SW simulation, but some works has covered this aspect [20]. Providing QEMU with the required SW simulation capabilities is an active research area [6, 44]. Recently, Imperas has released its SW emulation technology as Open-Source [26].

Native co-simulation is based on the direct execution of the source code on the host. Simulation speed can be improved avoiding the use of processor models at the expense of some estimation accuracy in order to enable an efficient system evaluation [18]. Estimation errors of about 30–40% can be accepted for initial system assessments [29]. Using either static [5, 9, 24] or dynamic techniques [11, 31, 46] the SW execution times are estimated and annotated to obtain timed simulations of the application SW. A simple version of this technique has been implemented in a commercial tool [27]. This kind of analysis technique has proven to be useful for power consumption estimation [10, 34, 48] as well. Native simulation has shown its ability to estimate the number of cache misses together with execution time [14, 57].

An integral part of the embedded SW is the RTOS used. In the two binary simulation technologies commented above (ISS and virtualization), the complete embedded SW including the application SW and the RTOS are compiled together. In native simulation this is possible whenever appropriate modeling of the Hardware Abstraction Layer (HAL) functions is made [44]. The main advantage of this technology comes from the possibility to easily include all the Hardware-dependent Software (HdS), such as RTOS, drivers, etc. Whereas the main disadvantage is that one needs to develop, in advance, the complete embedded SW including the application SW, the chosen RTOS, drivers, etc. An additional limitation is that each RTOS and each microprocessor requires a specific HAL model.

An alternative for native SW simulation is based on using an abstract OS model. As the RTOS functionality is abstracted, this approach is faster than the HAL-based one. Several alternatives have been proposed, from generic [19, 22, 23, 63] to real OS models [21, 47]. The former technology is currently supported by some commercial tools [16, 17]. The latter approach has the advantage that, when efficiently exploited, a more accurate model of the underlying RTOS can be achieved. This additional efficiency is obtained by combining the source-code execution time estimation with the OS model providing more accurate results. The work in [12] provides an analysis of the impact of including the OS time in the overall system estimation.

These abstract RTOS techniques also dedicate a large effort to accurately integrate time annotation and OS modeling with HW/SW communication, especially for HW interrupt management. Very few of these approaches support a real OS Application Programming Interface (API) such as TRON or POSIX [21, 47]. The idea of integrating facilities for OS modeling in SystemC was proposed as a new version of the language some years ago by the SystemC consortium. However, OS modeling at system level proved to be a much more complex task than expected, becoming an active research area.

However, none of these modeling methodologies are aimed at complex Multi-Processor Systems-on-Chip (MPSoC) modeling. Current MPSoC modeling requires dynamic task mapping, drivers and interrupt management which are not covered by previous modeling techniques especially with the requirement of fast simulation speed during performance estimations and system dimensioning. Moreover, a framework is required that supports a complete model of the platform that can easily integrate new components, either an application-specific HW or a programmable processor. When the application SW code of each function has not yet been developed, the underlying simulation technology supporting native simulation can be used for high-level performance analysis [16, 17, 36].

In this chapter, M3-SCoPE, a framework for performance modeling of multi-processing embedded systems for fast DSE is described. The proposed system simulation technology includes abstract models of the RTOS and the multi-processing architecture that can easily integrate the application SW through the RTOS API (i.e. POSIX). The SW is annotated with estimations of the execution time and power consumption. The multi-processing architecture is connected through an abstract Transaction-Level Model (TLM) of the bus with the peripherals and application-specific HW components. Different nodes in the system can be connected through networks. The SW simulation technology includes novel features such as dynamic time estimation and cache modeling.

Additionally, to allow evaluating complete, heterogeneous systems in an efficient way, simulation tools require capabilities to automatically create the system models for the different configurations. On one hand, DSE tools do not have the capability to drive the creation of the platform models. On the other hand, need for manual recoding for each evaluation point results in too long exploration times. Several previous works have been proposed to solve or minimize the effort required for platform model creation. Automatic generation of system models oriented to specific target architectures has been proposed in [35, 60]. Other works have been focused on automating the exploration of component interconnection [33, 42, 61]. However, none of these approaches cover model generation for complete architectures in the general case.

To provide more generic techniques, TLM techniques based on system-level design languages like SystemC have been proposed [13, 25, 43, 55]. Green-socks is an open source infrastructure for distribution of TLM models [32]. In [45] a TLM framework for automatic system model generation is proposed. The framework receives a fixed system description and generates the executable system model. In [39, 51] TLM infrastructures are used to accurately estimate SW performance.

Some commercial tools [3, 16] can model designs at TLM level. The schematic entry tools simply provide a graphical interface for plugging existing database models together. These models are described and connected at the transaction-level. They also provide shell interfaces which allow modifying the characteristics of the system components. However, the system architecture is fixed and cannot be modified. An alternative solution to schematic entries for system description and model generation is using XML based descriptions. IP-XACT [28] standard describes an XML scheme for meta-data documenting Intellectual Property (IP) used in the development, implementation and verification of electronic systems. This scheme provides a standard method to document IP that is compatible with automated integration techniques. Several tools have been developed to support that integration [37, 41]. The resulting models can only configure certain parameters on the system components. However, modifiable platforms cannot be described through IP-XACT and modeled with these tools. Thus, the exploration of the best platform architecture cannot be performed with these tools.

In that context, the work presented here proposes a solution to describe modifiable architectures and automatically generate the corresponding system models. These models can be configured by modifying the parameters of the system components and also modifying the system architecture itself. This modeling capability will allow the DSE tools not only to find the optimal tuning of the system components, but also to optimize the system itself. System descriptions will be performed in a simple XML format, although the proposed solutions can be easily adapted to other XML descriptions.

2.2 Native Co-Simulation Infrastructure for DSE

As stated above, SystemC has been widely adopted for system modeling during last years. SystemC provides features to describe systems from RTL descriptions up to system-level models, in order to enable its application during all the first steps of the design flow. Furthermore, as SystemC is a library of C++, SW designers can integrate their C/C++ codes together with the HW platform descriptions to create models of the entire system. Then, the SW is compiled and executed natively in the host computer without requiring slow ISS models. As a consequence, this solution results in very fast simulation technology.

However, this solution, which can be used to verify the system functionality, presents several limitations in terms of additional performance estimations. The native execution of a SW code provides no information about the temporal behavior of the code in the target platform. Furthermore, this kind of execution of the embedded SW does not consider any kind of processor allocation nor the effects of an operating system. Finally, the HW/SW interfaces are not adequately modeled, so the effects of interrupts, device drivers or bus contentions are not taken into account.

SCoPE is a library that provides the extensions required by SystemC to enable the use of native SW execution together with SystemC HW descriptions in order to

obtain performance estimation of the entire system. These extensions include facilities to accurately estimate and model the temporal behavior of the application SW [52], to emulate the behavior of the operating system [47] and to enable realistic modeling of the HW/SW communication [50]. As a consequence, SCoPE is a powerful infrastructure for high-level system modeling during early system performance evaluation.

Nevertheless, an efficient modeling infrastructure for DSE requires additional features. Exploring the effects of different component frequencies, variable number of processors or different memory architectures also requires the capability to describe and automatically build the system models during the exploration. Common DSE tools are developed to select the experiments to be evaluated and to extract the best points from the Pareto curves. However, these tools have no intelligence to create the different system descriptions to be simulated. Current DSE tools are not capable of selecting, instantiating and connecting the components of the SystemC model depending on the selected configuration.

To solve this limitation, M3-SCoPE combines SCoPE features with an input/output interface where XML-based descriptions of highly configurable systems can be provided [49]. From these descriptions the library automatically creates the system model by instantiating and connecting all the components with its selected parameters. This capability avoids manual recoding during the exploration process, increasing the overall exploration efficiency. Additionally, the use of XML files enables run-time system modeling creation. Thus, no recompiling steps are required during simulations.

Summarizing, the main benefits provided by M3-SCoPE for the designers are the following:

- Capability of describing configurable system descriptions in XML format
- Automatic system model creation at run-time, avoiding recompiling steps
- Performance estimation (time and power) and annotation of the application SW
  - Modeling cache effects
  - Modeling compiler optimizations.
- Modeling the effects of the operating system and processor allocation
  - Scheduling, synchronization and communication
  - Memory space separation.
- HW/SW communication
  - Using device drivers and interrupts
  - Direct access to HW registers through pointers.
- Facilities to automatically estimate performance features from the HW components, to be delivered to the DSE tools.

Considering all these features, M3-SCoPE can be easily integrated in the DSE flow (introduced in Chap. 1) as shown in Fig. 2.2. For this integration two input XML files are defined to provide the system description: one XML file describing the system and its configuration options, called *"System Description"* file, and another XML file of pairs identifier-value, fixing the selected configuration for each experiment, called

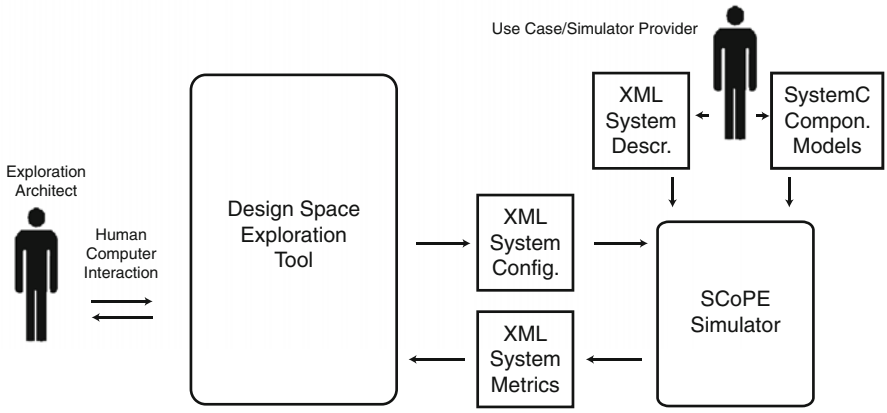

Fig. 2.2 Integration of the M3-SCoPE Simulator in the exploration flow. The system designer provides the configurable system description and the codes for all the system components. Then, DSE tool starts selecting configurations and receiving performance estimations from M3-SCoPE

"Configuration" file. An XML output file has been defined to return the simulation results. The external DSE explorer indicates the simulator which is the configuration to be analyzed each time by generating the corresponding *"Configuration"* file. The simulation tool interprets the file, builds the system model and performs the simulation. This means that no user interaction or model recompiling is required once the exploration process starts.

The tool generates an output file, when the simulation finishes. This file contains the values of the metrics obtained during the simulation. The output information is used by the DSE exploration architect to perform the search of the best solutions applying the RSM techniques.

2.2.1 Configurable XML System Descriptions

The XML System Description file includes information about the HW components, the HW architecture, and the SW tasks. A simple XML format has been developed to easily describe highly configurable platforms. The language guarantees fast model creation and efficient system simulation.

A simple example of an XML description using this language is shown in Fig. 2.3. To keep it simple, no configurable options have been added. The example proposes a system with a processor and a memory connected to a bus. A standard *"hello world"* application has been selected to execute in the processor. To describe a system with several platform architecture options and its configuration possibilities three XML mechanisms are provided by M3-SCoPE. All three mechanisms can be used simultaneously to describe highly configurable systems.

Fig. 2.3 Simple XML system description containing information about the HW components, its instances, the operating system, the SW applications and the task allocation. The description represents a simple system containing a processor, bus and memory, running a typical `"Hello world"` application

```
<HW_Platform>
  <HW_Components>
    <HW_Component category="bus" name="AMBA" frequency="200MHz" />
    <HW_Component category="processor" name="arm926t" frequency="200M"/>
    <HW_Component category="memory" name="Memory"
    mem_size="500MB" frequency="200MHz" mem_type="RAM" />
  </HW_Components>
  <HW_Architecture>
    <HW_Instance component="AMBA" name="my_bus" >
        <HW_Instance component="arm926t" name="my_proc" />
        <HW_Instance component="Memory" name="my_memory"
            start_addr="0x80000000" />
    </HW_Instance>
  </HW_Architecture>
</HW_Platform>
<Application>
    <Functionality>
        <Exec_Component name="hello" category="SW" function="hello"/>
    </Functionality>
    <Allocation>
        <Exec_Instance name="Hello_world" component="hello"
            processor="my_proc"/>
    </Allocation>
</Application>
```

2.2.1.1 Configuration of the System Components

The first configuration option is used to tune the characteristics of the system components. The values of all parameters in the XML file can be replaced by identifiers when the parameter is a configurable one. For example, the bus frequency that was indicated in Fig. 2.3 (`200 MHz`) can be been replaced by the identifier `"FREQ"`.

To select a configuration, the values of all identifiers must be assigned in the System Configuration File. Thus, to perform different simulations it is only required to modify the value-identifier pairs in the System Configuration file (Fig. 2.2). Applying this solution, the simulation of each experiment required by the DoE is performed by substituting the identifiers of the configurable parameters by the selected values and creating the corresponding system model.

2.2.1.2 Replication of System Components

The second configuration option is to indicate the number of times a system component is replicated. To do so, a new XML `"repeat"` clause is provided. This clause defines the number of times the element is repeated, an index identifier and the initial index value. Figure 2.4 corresponds to an extension of the system description in Fig. 2.3 considering that the number of CPUs in the system can be set by the `"CPUS"` parameter. This parameter must be assigned in the System Configuration file.

The `"repeat"` clause can be used to replicate both single components and groups of components, copying complete parts of the system architecture. If the value is set to '0', the element is not placed in the system. This option is used to add or delete different components within the system, including modifying SW components, HW components and the communication infrastructures. As a consequence, different platform architectures can be described.

Fig. 2.4 XML description with configurable number of processors. Depending on the parameter "CPUS", defined for each simulation by the DSE tool, the system model is created. Task allocation is modified depending on the number of processors

```
<HW_Platform>
  <HW_Components> ... </HW_Components>
  <HW_Architecture>
      <HW_Instance component="AMBA" name="my_bus" >
            <repeat number="CPUS" index="i" init="1">
                <HW_Instance component="arm926t" name="my_proc[%i]" />
            </repeat>
            <HW_Instance component="Memory" name="my_memory"/>
      </HW_Instance>
  </HW_Architecture>
</HW_Platform>
<Application>
      <Functionality>
            <Exec_Component name="hello" category="SW" function="main" />
      </Functionality>
      <Allocation>
            <repeat number="CPUS" index="j" init="1">
                <Exec_Instance name="task[%i]" component="hello"
                      processor="my_proc[%j]"/>
            </repeat>
      </Allocation>
</Application>
```

Fig. 2.5 XML description with configurable HW architectures. The figure shows a two possible configurations for as an interconnection component: a bus or a Network-on-Chip (NoC)

```
<HW_Platform>
    <HW_Components> ... </HW_Components>
    <HW_Architecture name="arch1">
        <HW_Instance component="AMBA" name="my_bus" />
        ...
    </HW_Architecture>
    <HW_Architecture name="arch2">
        <HW_Instance component="NoC" name="my_noc" />
        ...
    </HW_Architecture>
</HW_Platform>
<Application> ... </Application>
<Simulation>
      < Implementation HW_Architecture="ARCH" / >
</Simulation>
```

2.2.1.3 Selecting Complete Configurations

The third configuration option provided is to define several complete configurations and select one on each simulation. For example, in Fig. 2.5, two different HW architectures are described (`"arch1"` and `"arch2"`). The one selected for each simulation is defined in the `"Implementation"` clause. In this example, the architecture selected depends on the `"ARCH"` identifier. Its value must be set in the System Configuration file to `"arch1"` or to `"arch2"`.

The system description mechanism allows dividing the system description in parts and exploring different combinations. Multiple HW component lists, HW architectures or SW allocations can be described to be explored by the DSE tool.

2.3 Modeling of SW Components through Native Simulation

2.3.1 Performance Modeling of Embedded SW

As mentioned before, native simulation is a very fast solution for functional modeling of SW components. However, to obtain estimations of the system performance the untimed functional executions must be transformed into timed SW models. To do so, the technique applied in M3-SCoPE has two steps. First, performance estimations of the SW code are done statically, adding time annotations to the SW code. Then, the code is executed, and these times are applied to obtain a timed simulation.

The simulation time of the application SW is estimated depending on three elements: the code, the compiler modifications and the cache effects. Additional times required by the processors accesses to the rest of the HW platform (memory or peripherals) are evaluated inside the HW platform models, and are not directly estimated by the SW modeling.

To consider the three elements, the code annotation is based on a basic-block solution. For each basic block, the time required for its execution, the power consumption and the required cache accesses are annotated. Using this basic-block information, M3-SCoPE obtains, during simulation, the total time, power and cache accesses by accumulating the values for all the basic blocks executed. Additionally, estimation of basic block metrics from a cross-compiled binary code allows to account for the compiler effects.

The estimated execution time is then applied to transform the native SW functional execution into a timed simulation. SystemC time annotation is performed by means of `"wait (time)"` sentences at communication and synchronization points, which correspond to system calls and I/O accesses. However, the use of standard `"wait"` clauses does not allow modeling preemption in the SW tasks, as the time is completely applied independently of additional events in the system, as interrupts. To solve that, the technique of pre-emptable waits is applied.

The application of this extra information involves a penalty in the simulation performance. Additional instructions require additional simulation time, which is opposite to the main requirement of DSE simulators: speed. As a consequence, these annotations must be minimal. The solution of basic-block annotation minimizes the problem. The use of global variables to accumulate time and power also helps to reduce the overhead (Fig. 2.6). But probably the most important solution applied is to move the cache address searches from the cache models to the code annotation, as it will be presented later.

In parallel to time estimations, the proposed techniques are able to estimate the power consumption required by the processor to execute the application software. The approach consists of assigning an average value of energy to each machine instruction. By analyzing the cross-compiled code, the library obtains the number of instructions for each basic block, and this value is back-annotated in the original source code along with time annotations.

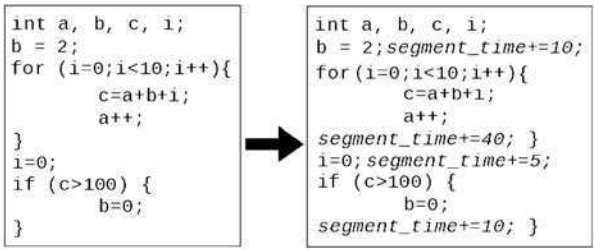

Fig. 2.6 Example of extra code for annotating delays. At the end of each basic block the global variable "segment time" is increased with the block delay. At system calls the final time value is applied to the simulation

When running the simulation, this extra code is executed so we can estimate the number of machine instructions executed. By multiplying this value by the average energy per instruction provided in the XML files, an estimation of the total energy required by the processor is obtained. It should be noted that, while this approach is valid for all processors, the accuracy of the technique depends on the stability of the power consumption of the target core.

2.3.1.1 Basic-Block Time Estimation

The estimation of the cost of each basic block in terms of time and power requires using cross-compiled code to take into account compiler optimizations. However, these compiler optimizations make it very complex to associate source code and generated binary code [1]. Complex reverse compilation techniques can be applied to correlate assembly blocks to source blocks [15]. However, this challenge is not always feasible due to advanced compiler optimizations.

In this work, a hybrid level technique for SW annotation is proposed: while basic block identification is performed at source level, characterization is obtained from assembly code. This strategy simplifies the characterization process and speeds up the analysis time.

To apply this solution to the DSE flow, a minimum effort from the designer is required. The cache model and its associated methods have been implemented as an independent library. Additionally, an automated instrumentation process performs all necessary annotations in source code to link software execution with the cache model. Figure 2.7 shows an overview of the cache estimation process, including basic block characterization.

Due to the rich syntax of source code, a C/C++ code parser has been developed in order to identify declarations, statements and expressions during the code annotation. The parser, based on a C/C++ YACC grammar, dumps the Abstract Syntax Tree (AST) to a file in XML format. This XML serves as input to the block identification and instrumentation processes. With the information included in the XML, the original C/C++ code is rebuilt adding time, power and cache information. Furthermore, this XML can be considered as an intermediate representation, independent from the source language. Any language could be estimated by simply creating a front-end for that language.

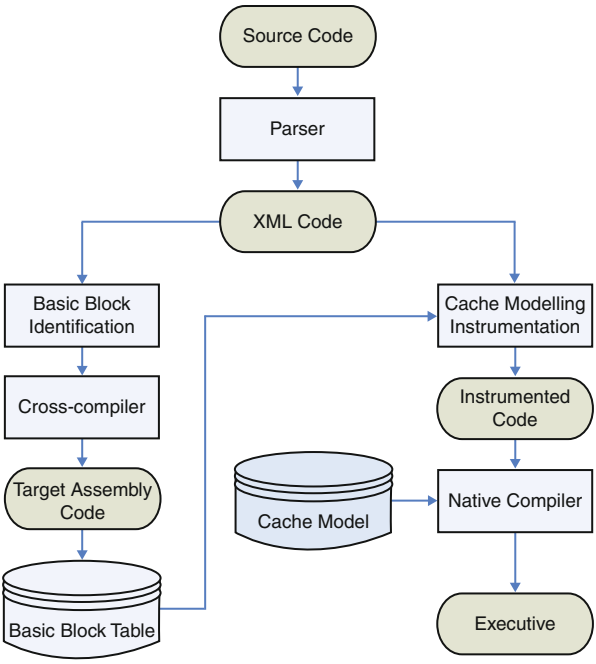

Fig. 2.7 Complete Annotation Process. The original code is analyzed with a C++ grammar. Then the code is built adding marks to identify the basic blocks and cross-compiled. The resulting code is analyzed to evaluate the metrics of each basic block, and the code is finally rebuilt adding this information

To identify basic blocks at source level, the proposed solution is to insert specific marks at the beginning and at the end of each basic block. This marked code is then cross-compiled, so the marks introduced are preserved in the target assembly code. The inserted marks looks like:

```
asm volatile("mark_xx:");
```

To prevent the compiler optimizations from deleting or moving such marks, they are declared volatile. Additionally, to keep the behavior of the original code, the asm instructions inserted consist simply of labels. This procedure guarantees that there is always a direct correlation between source and assembly blocks. However, as the effect of the compiler is constrained by the marks, the final binary code is not exactly the target one, so some errors are generated.

Compiler optimizations might affect both intra-block and inter-block behavior. Intra-block optimizations are considered in the characterization of the blocks from assembly code. This assembly code already includes both front-end and back-end optimizations. Inter-block optimizations are considered by delimiting the basic blocks at source level. Nevertheless, there are some compiler optimizations which cannot be accurately considered with this technique. Loop unrolling replicates the body of a loop statement in the assembly code, but from source point of view it is a unique block.

Nevertheless, in processors with instruction cache, loop unrolling is rarely employed, since the miss cost for each iteration significantly exceeds the jump cost.

Calls to inline functions from different points in the code have a similar problem, since the replicated code in the assembler file cannot be considered in the source instrumentation. If function inlining does not require replication of code (i.e. within a loop body), the technique works without limitations.

Summarizing, the error implied by this technique is minimal, and very adequate considering the accuracy required at the abstraction level.

2.3.1.2 Cache Modeling

Cache memories have also a very important influence on SW performance. Thus, cache modeling is required to obtain accurate estimations. As it is well-known, cache memories consist of lines that are arranged in sets, depending on the degree of associativity. This value determines how many lines are grouped in each set. As an example, the ARM920T [2] instruction cache is 512-line size with a 64-degree of associativity.

When the processor requires the information placed in a certain address, it is required to search in all the possible locations for that address. The number of locations depends on the associativity degree. For high degrees, this task is a time-consuming process, which is really critical in the abstraction level required for efficient DSE. An example is shown in Fig. 2.8. While the overhead of the time annotation is only one additional code line, the cache access requires a function call on each line to check the instruction address, and a complex search within each call to check if the values are in cache or not. This drawback is then the main focus of the cache model to speed up simulation time.

The solution proposed to minimize the simulation overhead is based on three main improvements: search cache lines instead of single instructions, replace a list search by a static annotation and move the search from the cache model to the source code. This is shown in Fig. 2.9.

As instructions within a basic block are sequential, it is not required to search for each address in the cache. If one value (instruction or data) is in cache, all the instructions/data in the same line will be also there. Thus, the number of checks is extremely reduced. If a common cache line contains 8 values, cache checks are performed only 1/8th of the times the actual cache is checked. In practice, lines

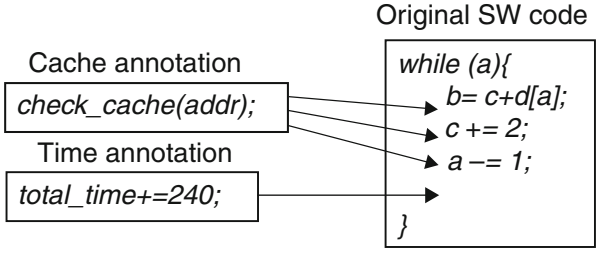

Fig. 2.8 Comparison between the annotation for modeling execution time and instruction caches. The different mechanisms result in different simulation overheads

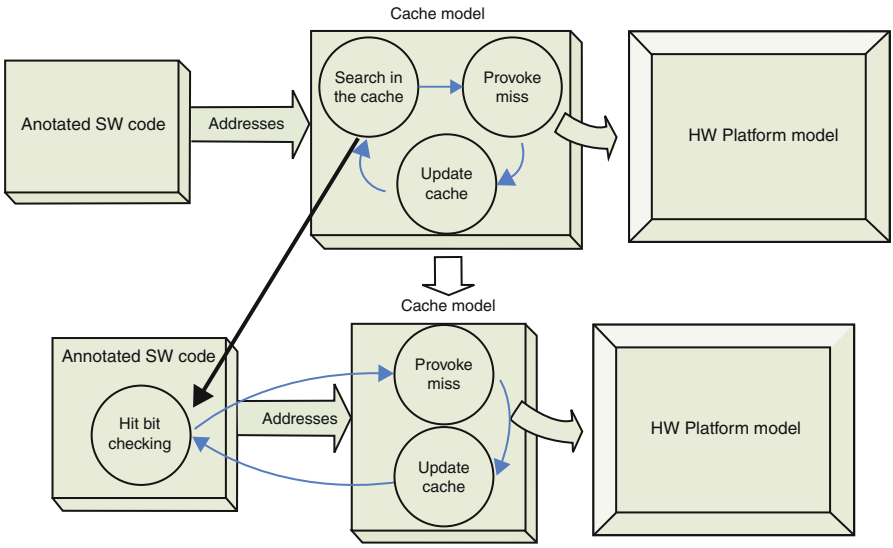

Fig. 2.9 The underlying concept applied to reduce of cache modeling overhead. The idea is to reduce the overhead by removing the function calls and searches done in the cache that are required when a new line is required. Instead, a bit checking is integrated in the software code annotation. Thus the cache model is only accessed at cache misses

containing instructions of different basic blocks are checked in all the blocks, so the reduction is slightly reduced.

The second improvement is even more important for complex caches. In order to avoid time-consuming *"tag"* searches, the dynamic search is replaced by a static *"bit checking"* solution. In the proposed model, a cache line is modeled as a structure which includes the target set at which it may be allocated and a flag which determines if a line is currently allocated in cache:

```
struct icache_line {
    char num_set;
    char hit;
}
```

The cache is modeled as a two-dimension array of pointers to cache line structures. The dimensions of the array depend on the physical characteristics of the cache to be considered (size, associativity and line size). All necessary methods for fetching or replacing data are also provided with the model. The caching mechanism consists of storing the addresses of the line structures in the array. An empty location in cache is modeled as NULL value.

The struct for each cache line is declared statically inside the SW code as part of the annotation, not in the cache model. This is done with the following line.

```
static icache_line line_124 = {0,0};
```

Thus, in order to check if the line is in cache or not it is enough to check the `"hit"` bit.

```
if (line_124.hit == 0)
     insert_line(&line_124);
```

This code is executed every time a basic block ends. The method used to model a cache miss, `insert_line()`, takes the address of the miss line and allocates it in cache. The allocation algorithm works as follows: the address of the line is used to find the target index. If there are free locations for this index, the address is simply recorded in the line pointers array and the hit flag of the line is set to true. If the index lines are full, the victim selection algorithm selects the victim line that must be expelled. The hit flag of the victim line is set to false through its pointer, and its location in cache is replaced by the new one. This way, the next time the victim line is required, the hit field will be false.

```
insert_line(icache_line *line) {
     icache_line *victim;
     victim = get_victim_line(line->set);
     if (victim != NULL) victim->hit = 0;
     line -> hit = 1;
     victim = line;
}
```

As a consequence of the entire process, tag searches are eliminated, and the cache is accessed only at misses, which is a small percentage of all the cache accesses. Thus, the cache modeling overhead is minimized. For data caches the solution applied is similar.

Note that no real caching of data is performed in our model during execution. As the SW code is executed natively, its execution does not depend on the behavior of the associated cache in the HW platform. Our cache modeling technique only models performance effects, not the actual HW behavior. This makes the technique faster than other cache simulation approaches.

2.3.2 RTOS Modeling

One of the most important points when transforming a high-level description into a real SW implementation is the inclusion of the OS. The SW code is gradually refined, mapping high-level facilities, such as communication and synchronization facilities, to the target OS facilities. Thus, the development infrastructure has to support the refined SW code, providing the target OS facilities inside the timed TLM model of the application SW.

The OS model included in M3-SCoPE is divided in a similar way to real OSs. An OS can be divided into several layers [59]. In the upper part, we can find the system call handlers that support the API. In the middle, the layers for internal OS management form what can be called the kernel core. At the lower level, there are

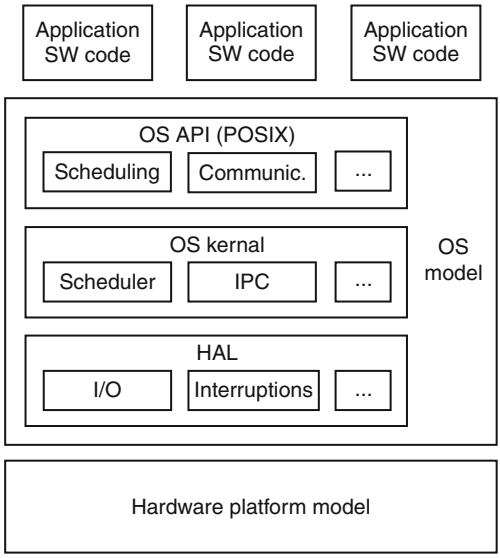

Fig. 2.10 Architecture of the OS model. The model is divided in three parts: the upper part contains the implementation of the services directly called by the application software. The middle part contains the model kernel, where all the internal OS services are implemented. Finally, the lower part is in charge of enabling the interconnection of the software with the rest of the simulation

the layers oriented to hide the HW platform details (Hardware Abstraction Layer, HAL) from the rest of the system, such as low-level functions to access the bus, the interruption management or the context switch infrastructure. Thus, the proposed OS model will provide support for all of these common OS sections, as can be seen in Fig. 2.10. In order to simplify the models, all the possible layers have been grouped into three layers, following the classification above. To model the execution time of the OS itself, a delay is associated to all system calls and some internal functionalities, such as context switches or interrupt handlers.

The kernel core is mainly in charge of process and thread execution management, memory management and task communication and synchronization. To model task scheduling characteristics, a new scheduler is placed on top of the original SystemC scheduler. The OS model scheduler controls task execution by maintaining only one unblocked thread per processor. This scheduler manages the priorities and policies as indicated in POSIX, integrating a two-level scheduler: process scheduling and thread scheduling. The scheduler is called when a task is blocked or unblocked as an effect of a system call, or when an interrupt resumes a task preempting the current running task.

This functionality has been implemented with use of the SystemC features. Each thread in the application SW is associated with a SystemC thread (`SC_THREAD`) running the corresponding code. In this way, the SystemC scheduler is in charge of managing the thread context switches. To model the OS thread scheduling the SystemC scheduler is hidden. All the SystemC threads are suspended with SystemC `wait` statements. When the scheduler selects one of these threads, a `notify` call is performed to awake the corresponding thread. When the thread is preempted or blocked, it returns to the `wait` statement and a new thread is scheduled. Thus,

the SystemC scheduler executes the only one runnable thread each time. This way scheduling is completely managed by the OS model scheduler through `wait` and `notify` statements. Note that SystemC `wait` statements are used for two purposes: thread blocking for scheduling purposes, as explained here, and for time annotations as explained in the previous section.

However, SW code refinement implies more than optimizing task scheduling characteristics. The proposed OS model provides a wide set of OS facilities, in order to allow the transformation of high-level descriptions into real executable SW codes. High-level features for communication, synchronization or time management have to be substituted by real OS features. In our proposal, a POSIX API has been implemented for communication (e.g. message queues or sockets), synchronization (e.g. semaphores, mutex or conditional variables), signals, timers, sleeps and many other functionalities. Thus, the designer can simulate the real SW code together with the rest of the system before moving to more accurate but time consuming simulation infrastructures, such as ISS-based simulations.

To support several independent OS models the OS model information has been encapsulated in a C++ class. Thus, modeling several OSs only require instantiating an OS class several times. Multiprocessor management is supported by maintaining as many running threads as processors. When a process is created or preempted, the presence of an idle processor is verified. When a processor is delivered, a new thread is scheduled to run on this processor. In this way, all processes are maintained active, and tasks are scheduled as soon as a processor becomes empty.

As all threads are SystemC threads, and the SystemC kernel activates the threads sequentially, there is no real parallelism during the simulation. Parallelism is only emulated. As a consequence, no additional mechanism is required to ensure safe concurrency inside the OS model, in contrast to real Symmetric Multi-Processing (SMP) OSs, where mechanisms such as *spinlocks* are critical to handle concurrency safety. Furthermore, the OS modeling technique is flexible enough to cover the most usual OS architectures. Each OS is associated to a memory space independently of the underlying processing architecture (bus or network). The complete MPSoC may contain as many memory spaces as needed. The HW platform model creation presented above easily enables that feature.

More complex is the modeling of memory space separation among processes of the same OS. As the SystemC simulation is a single host process, the management of memory spaces in the simulation requires additional mechanisms. Reusing names of functions and global variables among component codes or loading several instances of the same task is not possible without a memory space separation infrastructure. Nevertheless, this problem is not limited to processes of the same OS. As the entire simulation is a single host process, name duplication between SW process of different nodes, running in different OS, or between SW and HW components provokes the same error. A general solution has been implemented in M3-SCoPE. It will be presented hereafter when describing how all the system components are integrated in the SystemC model. Finally, the technology has also proven to be accurate enough to model dual General-Purpose/Real-Time Operating Systems, such as RTLinux. More information can be found in [52].

2.3.3 Modeling of SW/HW Communication

One of the most important features in native co-simulation is the interconnection between the native SW execution and the HW platform model. All accesses from the SW code to the HW platform must be detected and redirected, otherwise the SW will attempt to access the host peripherals, provoking multiple failures.

2.3.3.1 HW/SW Communication Using Device Drivers

The OS kernel and the API presented above are not enough to provide a complete OS model for software modeling and refinement. Although application code refinement is mostly supported, Hardware dependent Software (HdS) (such as drivers) is not supported. A more complete model, capable of managing interruption handlers and drivers, is required. The HdS is usually critical in embedded systems and its analysis at the early stages of embedded system development can provide large benefits during the rest of the project.

In order to perform correct HW/SW interface modeling it is required to have an OS model capable of managing the I/O accesses from the processor to the peripherals, and the interrupts the peripherals send to the processor. To manage the I/O, Linux-based driver support is provided. Linux standard functions for registering, accessing and managing drivers are integrated in the OS model, allowing the creation and use of Linux drivers.

Interrupts are generated from the HW platform model and received by the application SW modeling infrastructure, as explained in the previous section. When the timing effects have been established, the OS facilities for interrupt handling are called by the SW modeling infrastructure, calling the corresponding `"wait"` and `"notify"` statements. The standard or user-defined handlers associated with the interrupts are executed, performing the proper operations, and calling the corresponding drivers. The HW timer interrupt is of special interest. It is used to manage timers, sleeps, alarms and time-based scheduling policies, such as Round-Robin.

That way, HW/SW communication explicitly indicated in the code can be modeled. The I/O functions connect the native SW execution with the HW platform model. However, in embedded systems, HW/SW communication is not always so explicit. Peripherals can also be accessed by reading and writing directly the addresses of the peripheral registers. This communication is possible by using C/C++ pointers.

2.3.3.2 Modeling of Direct I/O Accesses Through Pointers

When the addresses of the peripherals are known by the SW developer and the OS allows that, peripheral registers can be directly accessed. For example, a device connected to a serial port can be accessed in the following way:

```
char *uart_addr = 0x80001004;
*uart_addr = A;
```

However, when the code is natively executed, this access will not work. Firstly, the host computer does not physically contain the required peripheral. As these pointer accesses cannot be easily detected statically, additional mechanisms are required to detect and redirect these accesses to the HW platform model at run-time.

Secondly, in native co-simulations the operating system prevents the application code to access specific HW addresses. When the SW code tries to access a fixed HW address, the memory management unit (MMU) will raise an exception as there is no physical address associated with the required virtual address. In an OS such as UNIX, this will result in a segmentation fault. The way to solve this problem is to force the operating system to create a page of virtual memory at the desired memory address. Thus, when the SW under simulation wants to read or write the HW values, values are correctly stored in the host memory.

To force the native operating system to create this memory page, the standard POSIX `"mmap"` function can be used. The `"mmap( )"` function shall establish a mapping between a process address space and a file, shared memory object, or typed memory object. The format of the call is as follows:

```
pa = mmap(addr, len, prot, flags, fildes, off);
```

The required code is shown in Fig. 2.11. In that code, a file is created to store the information of the associated memory. It is important to note that the maximum size of the mapped memory is equivalent to the size of the associated file. As a consequence, if an empty file is used, no values can be read or written. The solution applied is to assign a size of `"len"` to the file before calling `"mmap"`. To do so, the standard POSIX function `"ftruncate"` is used.

If the initial address does not correspond to the beginning of a memory page, special management is required. Memory pages always start from an aligned position. Thus the memory activated will start at the corresponding aligned address and will cover `"len"` bytes. To adjust the addresses, there are two possibilities. First the `"offset"` parameter can be used to indicate where exactly the mapped memory area must start. The second solution is to increase `"len"` with the offset of `"addr"`. In the proposed code (Fig. 2.11) the second solution has been used. Furthermore, for debugging purposes, it is interesting to note that the values stored in the scratch-pad memory model can be shown by reading the associated file.

The solution is very effective when modeling scratch-pad memories in native co-simulations. It automatically allows executing SW code using fixed HW addresses with a negligible simulation overhead. As no cache misses or any other event is provoked internally by scratch-pad memories, only the ability of reading and writing values at these addresses is required. More specific details of internal scratch-pad operations are not handled at high level.

```
void initialize_periph (void *addr, int len){
 fd = open("tmp.txt",O_CREAT|O_RDWR,0x01b6);
 ftruncate(fd,get_page_size());
 len += addr - page_aligned(addr);
 mmap(addr,len,PROT_READ|PROT_WRITE,MAP_FIXED,fd,0);
}
```

Fig. 2.11 Code for mapping the HW memory model

Mapping HW peripheral accesses to a created memory page in native memory space results in another issue. These peripheral accesses from SW usually communicate particular event notification to the HW. Hence these HW peripherals must be notified about these HW access events. HW Peripherals are not designed to make polling of any variable. They react to read or write accesses from the system processors. Even in case of using a high-level model for the HW peripherals, it will also be based on such event-based communication. Whereas in technique presented in the previous section, the access is performed but the peripheral does not receive any event informing that a read/write operation has been performed in their registers.

The only way to produce the event is not to map HW access to a created memory page and let the simulation crash. When the simulation is going to crash due to the segmentation fault, the error can be captured, solved and then the simulation can continue. The advantage of this approach is that the HW access is detected and the event can be sent to the peripheral model. When the SW tries to access an invalid memory address, the native operating system raises a SIGSEGV signal. This signal can be captured with an appropriate signal handler. This prevents the program to terminate, but this solution creates another challenge. During the signal handler, neither the access type (read/write) nor the value are known. Hence, the HW access cannot be performed at the signal handler.

To obtain the data, the memory remapping technique presented in the previous section is used. At the signal handler the memory mapping is activated and the code returns to repeat the pointer access. To perform a correct read access, the read transfer to the virtual platform is done first, updating the necessary memory address. Thus, when retrying the pointer read, the value obtained is correct. To perform a correct write access, the pointer access is performed first, and after that the new value is sent to the virtual platform.

Performing an `"mmap"` allows retrying the instruction, but once an access has been performed, the memory page is active and further accesses are not detected. To solve that, the memory page must be unmapped. However, when the code returns from the failed instruction, it continues normally without unpin the page. A possible solution is to create a parallel thread that wait for a certain time and then unmaps the page. However, this is an unsafe solution. There is no guarantee that there will be no more accesses before the unmap step, and even there is no guarantee that the unmap is done once the application SW code continue the native simulation.

To unmap the memory page properly, the SW code itself must do it. Just after the memory access is performed, the page must be unmapped. To do that, the original SW code must be modified. The solution applied is to dynamically inject code after the load/store assembler instruction that provoked the error. This injected code disables the memory page, re-establish original SW code and continues the execution. As a consequence, when the HW access is performed, the peripheral model is informed and the simulation status returns to the correct point to detect new accesses. Although the memory page is unmapped the data stored are not lost. The values are saved in the file associated to the memory page. The entire process is summarized in Fig. 2.12.

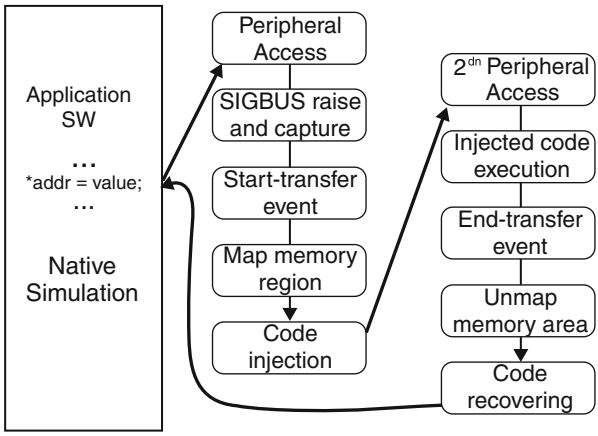

Fig. 2.12 Process for complete handling of HW accesses directly using pointers

Furthermore there is an another issue with the solution above. Detecting if a pointer access is a reading or a writing one is also complex. A possible solution is to disassemble the binary code of the instruction provoking the error, but this solution is not portable. Moreover, in x86 processors both reading and writing accesses are performed with `"mov"` instructions, so it is not easy to distinguish both. The portable solution is to force the system to raise different signals for read and write accesses. When executing an I/O pointer access, a SIGSEGV signal is obtained if the memory address has not been mapped. If the address has been mapped but the associated file has 0 size, a SIGBUS signal is raised. Thus at initialization the address is only activated for reading accesses with an empty file. Thus, a SIGSEGV raises at writing accesses (there is no writing permission) and a SIGBUS raises at reading accesses (there is no area in the associated file). This way one can differentiate between read and write HW peripheral accesses. More details on this scheme can be found in [50].

2.4 HW Platform Modeling

Once the software modeling has been defined, it needs to be connected to a HW platform for HW/SW co-simulation. For this purpose, an intermediate infrastructure is proposed. This infrastructure is placed in the OS model and in the target platform modeling facilities.

The SW modeling facilities presented above in Sect. 2.3 require a bus and, sometimes, a network interface to simulate a real communication of the SW with the rest of the platform. The bus model has to interact effectively and efficiently with the HW, thus a simple and flexible HW interface is integrated. This interface allows connection not only to generic HW components provided by the tool, but also to user-provided HW modules written in SystemC. For that purpose, three auxiliary HW models are needed to model a complete communication framework: a network

interface for node interconnection, a DMA for large data transfers and an abstract model of the physical memory to exchange data with the rest of the platform.

2.4.1 Bus Modeling

The techniques explained for SW estimation are always focused on fast performance estimation. TLM models are very efficient for this kind of applications. In order to take advantage of the simulation performance that TLM can provide, Programmer View with Timing (PVT) models are used instead of bus cycle-accurate models. These PVT models use behavioral descriptions with simplified communication mechanisms to perform fast simulations.

The bus model developed uses the TLM library for fast data transmission. In this way, communications are modeled by considering complete payload transfers instead of word-by-word transfers. The bus manages the simulation time of each transfer by considering parameters such as bus bandwidth, packet priority and size. However, the bus needs another component to complete its functionality – an arbiter. This module performs the transfer scheduling based on packet priority and delivery order. Therefore, the threads that try to access the bus and are not scheduled are blocked in a queue.

Another feature to take into account is the possibility to create bus hierarchy to cover more platform designs. The bus interface is able to connect not only buses with HW models, but also with other buses. This is implemented making use of the standard TLM interface to communicate different components. This interface can interconnect several modules transparently to the users.

2.4.2 HW Interfaces

M3-SCoPE provides several generic HW components, such as memory, bus, network or DMA. Additionally user-defined components can be integrated in the HW platform. In order to facilitate the integration of these user defined components, a base class is provided to be used as a wrapper. This wrapper base class is in charge of handling the TLM2 bus protocol management. The user needs to implement only the functionalities for read and write functions.

Additionally, the wrapper includes an automatic power estimation infrastructure. For each component it is possible to define parameters for the static power consumption and dynamic power consumption for read and write accesses. For read/write accesses it is possible to define two energy consumption values, one dependent on the number of accesses and the other one that considers the size of the data transfers. Applying all these parameters, M3-SCoPE automatically obtains an estimation of the component power consumption to be added to the consumption of the rest of system components. As all the HW component models (user-defined models and

components provided by M3-SCoPE) inherits this base class, the XML interface can automatically obtain power metrics of all the system components, and send that information to the DSE tool without manual intervention.

2.5 Automatic Generation of System Models

To simulate each one of the configurations selected by the DSE tool, different system models must be created. The model descriptions can be obtained by applying the values of the XML System Configuration file to the XML System Description. To generate the system model, the simulator dynamically creates instances of the required models and builds a platform model by interconnecting them as specified in the XML files. The complete process is shown in Fig. 2.13: first, the SW code of each process and the SystemC HW descriptions are compiled creating dynamic libraries. Then, the simulator loads these libraries and creates the system model. Finally, the model is simulated and performance results are obtained.

2.5.1 HW Platform Creation

To dynamically create the HW platform model, it is required to instantiate first the selected components, and then to create the interconnections between them. To do so, configurable models of the HW components are required. Response times, delays, area, mean power consumption, power for access, frequency, memory size,

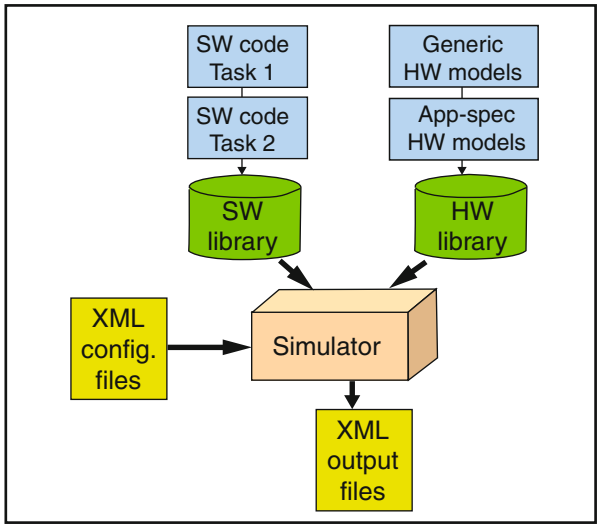

Fig. 2.13 Flow proposed for automatically creating the design models. First the SW codes and the codes describing the functionality of the HW components are annotated and compiled creating binary libraries. At simulation time, the simulator receives the system description and uses the corresponding codes from that libraries to perform the simulation

IRQ or associated memory map addresses are some of the configuration possibilities. To instantiate a component in the system model, all these parameters must be set. Parameter values are obtained from the values indicated in the corresponding `HW_Instance` clause of the XML System Description file (Fig. 2.3). These parameters can be either defined as explicit values ("200MHz", "500MB") or identified as configurable values.

To set the parameters which are not specified in the `HW_Instance` clause, the simulator checks the `HW_Component` clause corresponding to the type of component instantiated (Fig. 2.3). Similar to the previous ones, these parameters can be fixed or configurable ones. Finally, if any of the parameters has not been fixed, the default values for this component model are applied.

Additionally, both the generic models provided by M3-SCoPE and the models provided by the users are provided as Dynamically-Linked Libraries (".so" files). Using the host functions for dynamic loading, the components can be instantiated only requiring their names. After that, the instantiated HW components must be interconnected to create an executable system model. To simplify the interconnection work, TLM techniques have been used. TLM accurately describes the system communication architecture down to the level of individual read and write transactions. The use of transfers instead of signals reduces the complexity while automatically interconnecting the system components. To allow easy automatic interconnection of the system components, all component models have been created by using a generic template provided by the simulation engine. This template is oriented to ensure interface compatibility without limiting the component communication requirements. Ensuring that both ends of each interconnect have compatible interfaces, an automatic connection is possible.

Finally, to complete the HW platform generation, it is required to create the memory maps and to ensure correct interrupt delivering. Each time a component is connected to a bus, its associated memory area is integrated in the memory map, ensuring that it has not been used before. The solution is similar for network communication where network models require the node identifier in order to configure the internal routing protocols properly.

2.5.2 SW Components Instantiation

To create the SW infrastructure, OS models and SW tasks are finally added to the simulation as described in the XML files. OS models are provided by M3-SCoPE, while the application SW code must be provided by the user. OSs are mapped to a processor or group of processors (for SMP systems). SW tasks are associated to an OS, and thus mapped to a processor.

To integrate SW tasks in the simulation, SW code is annotated and compiled building a library which is added to the simulation. The annotated code provides the performance information required by an external DSE tool to perform the exploration.

Software tasks are defined in the XML System Description file indicating the name of the main function of the task. To load the main function, dynamic library management provided by OS is used, by calling the `dlopen` and `dlsym` function. Additionally, other parameters like the OS where it will run, priority, policy and the main function arguments can be defined. All these elements can be parametrized, so the DSE flow can explore the best configuration for the SW tasks.

2.5.3 *Integration of Independent Component Codes*

During the system model creation, multiple HW and SW components are integrated in a single SystemC simulation. When integrating different components in a system model, the models of each component can be incompatible for simultaneous integration in a single executable. If component models contain global variables and functions, name duplication problems can appear. This is especially important when integrating SW codes or algorithmic codes, where global variables are really common. If two components contain elements with the same names or a component is instantiated more than once, it will cause linking errors during compilation. Recoding of the system components can solve the problem, but it results in a very costly and error prone task.

To allow reusing names, two direct solutions have been considered in M3-SCoPE [53], considering *"direct"* as solutions without requiring recoding or any additional design effort. The first solution is to declare all functions and global variables as `static`. That way, the names are local to the object *".o"* files. Then, when linking all the object files generated by applying this solution, the linkage is correct.

However, this solution presents four main limitations. These limitations will be overcome during next sections applying different approaches. The main problems are:

- First, it requires recoding, as it is required to add the `static` qualifier before the declarations. Thus, it cannot be called a *"direct"* solution.
- Second problem is that this solution is only applicable if each component is isolated in a single object file. If the system component description is divided in several object files, the functions and variables are not visible by the rest of the component object files. Considering that dividing functionality of SW processes and algorithms in C/C++ in several object files is usually considered a good programming practice, the solution is not recommended.
- Third problem appears when a component is instantiated several times. If a component using global variables is instantiated more than once, all instances will share global variables. This will generate interferences among the instances and will end in wrong system operation.
- Fourth problem is that this solution can only be applicable to internal functions and variables. For example, if we want to integrate several SW processes, all starting

with a `main` function, it is not possible to declare them `static`. Otherwise, the functions are hidden to the simulation kernel, and cannot be called.

Thus, a different solution avoiding these problems has been implemented. The first two problems can be solved by forcing separate visibility scopes for functions and variables on each component. The third problem requires modeling different memory spaces. Defining different scopes is not enough to share variables among different instances. Finally, the fourth problem can be overcome by providing an automatic way of defining and loading different entry points with the same names.

2.5.3.1 Achieving Separate Visibility Scopes

Visibility scopes can be defined by creating *namespaces* or *classes* in the code. However, these solutions are limited and require recoding. The solution recommended to implement the separate visibility scopes is to create shared libraries with all the application SW codes required for each SW process to be simulated. Shared libraries are libraries that are loaded by programs at run time. As the library is not copied into the executable file, the executable size is minimized.

Dynamic libraries can be created using the following commands. In the code, two source code files are compiled to create the corresponding object files, and then both object files are integrated to create a shared library file:

```
gcc -fPIC -c filea.c #create object file
gcc -fPIC -c fileb.c #create object file
gcc -shared -o libmylib.so filea.o fileb.o
```

Shared libraries can be used in a static or a dynamic way. When used statically, at link time, the linker searches through available libraries to find modules that resolve undefined external symbols. Then, the linker makes a list with the libraries required by the executable. When the program is loaded, start-up code finds those libraries and maps them into the program's address space before the program starts. The instruction used to create the executable is the following:

```
gcc main.c -o executable.x -L($Library_Path) -lmylib
```

When used dynamically, the library is unknown at link time. The library is selected and opened during the execution of the program. As a consequence, the linker command does not require a `"-l"` flag with the library names and paths.

To execute a program requiring shared libraries, the library must be in the library path. When using a library created specifically for an example, it is expected not to be in a standard library recognized directory. To add the library to the search list, the environment variable `LD_LIBRARY_PATH` must be updated properly.

To create separate visibility scopes, shared libraries can be used in both ways. The specific compiler options required for that purpose are related with the library creation, not with its use. By default, a function of a shared library always calls the symbol of the main program if it exists. To create separate scopes, the scope of all library elements (functions and variables) must be defined as local to the library.

Thus, when a function in the library calls another function or global variable, it uses the elements inside the library, instead of accessing the elements of the main program. Only when a name is not resolved inside the library, the main program or other shared libraries are accessed. To force the program to use the internal library elements, the library visibility has to be declared as `protected`. To do so, when creating the library, it is required to specify the compilation flag:

```
-fvisibility = protected
```

Using this flag, it is only required to indicate the entry points of each component description in the XML files to enter the visibility scopes. Once the starting function of the correct library is entered, all internal operations of the component description will be isolated within the library, and without having or producing external interferences.

2.5.3.2 Modeling Separate Memory Spaces

Providing different visibility scopes guarantees integrating independent component descriptions in a single executable. However, if a component is instantiated twice, global variables are shared. In real SW implementations, the memory space separation provided by the OS for each process solves the problem. Thus, the simulation framework must provide an equivalent solution.

Using different host processes to create the SystemC simulation is a possible solution to achieve that. However, inter-process communication mechanisms, and additional context switches required to perform the simulation makes the solution a bit complex to use.

A more easy solution to solve the problem of multiple instances is to create a copy of the shared library for each instance. It provides an automatic solution which is easy to implement. Storage requirements are increased as several copies of the files are created. Nevertheless, disk sizes of host computers are usually large enough.

Using dynamic links (`ln -s`) instead of copies is not a valid solution, as the linker uses the final node directly.

2.6 Simulation Results

As a simulator oriented to evaluate different configurations of software centric systems, the most important parameters required to demonstrate the validity of M3-SCoPE are estimation accuracy and simulation time. The estimation accuracy of the simulator covers two main aspects: the evaluation of the performance of the software code and the modeling of the effect the software execution has in the rest of the system. The first aspect can be checked by comparing the time estimated by the tool and by an ISS (e.g. Skyeye [56]). The second aspect can be evaluated verifying the amount of data injected by the software modeling in the system buses, which means checking the number of cache misses estimated by the tool and by the ISS.

	Instructions			*Time*		
	Skyeye	***M3SCoPE***	***Error %***	***Skyeye***	***M3SCoPE***	***Speed-up***
Bubble	5200180007	5200180007	0	4m45,53s	3,76s	x71
Factorial	2747041	2996535	9,1	1,28s	0,02s	x64
Hanoi	18481575	17695142	4,3	11,15s	0,11s	x100
Coder GSM	13466069	14066581	4,4	10,6s	0,09s	x117
H264	5601674012	5800347641	3,5	5m25,6s	3,92s	x80

Fig. 2.14 Analysis of accuracy and speed of the estimation technique. The results obtained with M3-SCoPE have been compared with SKyeye [56], a no cycle-accurate ISS; the faster type of processor simulator

Example	**Instruction Cache Misses**			**Data Cache Misses**		
	M3-SCoPE	***ISS (SkyEye)***	***Error (%)***	***M3-SCoPE***	***ISS (SkyEye)***	***Error (%)***
Bubble	25	27	8	5204878	5199772	0.09
Hanoi	20	18	10	45	38	15.55
Factorial	8	7	12	500	375	25
GSM - 1 frame	3738	3663	2	660	670	1.49
GSM - 4 frames	15324	14814	3.4	2370	2452	3.46
GSM - 7 frames	26333	25730	2.3	4104	4235	3.19
GSM - 10 frames	37425	36624	2.2	5915	6026	1.84

Fig. 2.15 Accuracy of the cache models. The error is more representative in large examples, since in small examples, the number of errors is too small for comparisons

As can be shown (Fig. 2.14), the time estimation error of the native co-simulation technique is lower than 10%. Regarding the errors in the estimation of the number of misses, which has a high importance in bus and network modeling, the errors are in a similar range (Fig. 2.15).

In benchmarks with a small number of function calls, (i.e the bubble) the miss rate estimation errors is less than 1%, but when the number of calls to function is increased the miss rate grows too. Hanoi benchmark has a large percent of function calls in his instructions, his miss rate estimation error is 15% and in Factorial, which is an extreme code that only has function calls, the miss rate estimation error is 25%. This error is due to the big number of function calls, the difference in the compilers for different architectures produces that the number of registers saved in a function

were different. This error is important in specific cases, but in a larger codes it is minimized, as shown in the GSM coder example.

To demonstrate the modeling capabilities of all the presented features an the integration of the tool in a complete DSE process, an entire chapter is provided (Chap. 7), where M3-SCoPE is used to model a Power-Line Communication infrastructure.

2.7 Conclusions

SystemC has proven to be a powerful language for platform modeling. Nevertheless, its full exploitation for this purpose requires significant research activity in order to cover all the modeling requirements. The SCoPE framework has been described taking advantage of the language capabilities for system modeling and simulation. A methodology and associated library has been developed providing a complete OS functionality that can produce accurate timed simulations of the SW components. Power and timing estimations of the application SW running on the multi-processing platform in close interaction with the rest of the platform components can be obtained from the system simulation.

This chapter shows that native co-simulation can be applied successfully to this task. The performance analysis technology is fast enough to support efficiently the multiple runs required by DSE processes. To achieve this goal, M3-SCoPE has been developed with several improvements to SCoPE: cache modeling, direct I/O communication through pointers, memory space separation and dynamic platform creation from XML files.

This tool has been assessed on an industrial demonstrator, as shown in Chap. 7. The results demonstrate that the native co-simulation techniques proposed constitute a sufficiently accurate technique that is much faster than ISS-based simulation systems. This speed increase makes the technique optimal for fast system evaluation, covering the need to obtain effective metrics, which is necessary for efficient Design Space Exploration.

References

1. Aho, A. V., Lam, M. S., Sethi, R. & Ullman, J. D.: Compilers: principles, techniques and tools. ED Pearson, (2007).
2. ARM, Advanced RISC Machines Holdings. Retreived form www.arm.com
3. ARM Realview Development Suite (2005). Retreived form www.arm.com/products/DevTools/RealViewDevSuite.html
4. Bailey, B., Martin, G., & Piziali, A.: ESL Design and Verification: A Prescription for Electronic System Level Methodology. Morgan Kaufmann (2007).
5. Balarin, F., Giusto, P., Jurecska, A., Passerone, C., Sentovich, E., Tabbara, B., Chiodo, M., Hsieh, H., Lavagno, L., Sangiovanni-Vincentelli, A. & Suzuki, K.: Hardware-Software Codesign of Embedded Systems: The POLIS Approach. Springer (1997).
6. Becker, M. Di Guglielmo, G., Fummi, F., Mueller, W., Pravadelli, G. and Xie, T.: RTOS-Aware Refinement for TLM2.0-based HW/SW Designs, Proc. of DATE, IEEE (2010).

7. Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F., & Poncino, M.: SystemC cosimulation and emulation of multiprocessor SoC design, Computer, V.36, N.4, IEEE (2003).
8. Benini, L., Bogliolo, A., Menichelli, F. & Oliveri, M.: MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. Journal of VLSI Signal Processing, V.41, N.2: Springer.
9. Bouchhima, A. , Gerin, P. & F. Petrot: Automatic Instrumentation of Embedded Software for High Level Hardware/Software Co-Simulation. Proc, of ASP-DAC. IEEE (2009).
10. Brandolese, C., Fornaciari, W. & Sciuto, D.: A Multi-level Strategy for Software Power Estimation, Proc of ISSS, IEEE (2000).
11. Brandolese, C., Fornaciari, W., Salice, F., & Sciuto, D.: Source-level execution time estimation of C programs. Proc. of CoDes, IEEE (2001).
12. Brandolese, C. & Fornaciari: Measurement, Analysis and Modeling of RTOS System Calls Timing,11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, 2008.
13. Cai, L. and D. D. Gajski: Transaction Level Modeling: An Overview. In Proc. of CODES + ISSS03 (2003).
14. Castillo, J., Posadas, H., Villar, E., & Martinez M.: "Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-Simulation". Proc. of GSLVLSI, ACM (2010).
15. Cifuentes, C.: "Reverse Compilation Techniques". PhD thesis, Queensland University of Technology (1994).
16. CoWare: Task Modeling and Virtual Processing Unit User's Guide (2009).
17. Cofluent: CoFluent Studio: System-Level Modeling and Simulation Environment (2009).
18. Gerin, P., Hamayun, M. and Petrot, F.: Native MPSoC Co-Simulation Environment for Software Performance Estimation. Proc. CODES+ISSS. ACM (2009).
19. Gerstlauer, A., Yu, H. & Gajski, D. D.: RTOS Modeling for System Level Design, Proc. of DATE, IEEE (2003).
20. Gligor, M., Fournel, N. & Petrot, F: Using Binary Translation in Event Driven Simulation for Fast and Flexible MPSoC Simulation. Proc of Codes+ISSS, 2009.
21. Hassan, M. A., Yoshinori, S. K., Takeuchi, Y. & Imai, M. RTK-Spec TRON: A Simulation Model of an ITRON Based RTOS Kernel in SystemC. Proc of DATE, IEEE (2005).
22. Hassan, M.A., Sakanushi, K., Takeuchi, Y. & Imai, M.: Enabling RTOS Simulation Modeling in a System Level Design Language. Proc. of ASP-DAC, IEEE (2005).
23. He, Z., Mok, A. & Peng, C.: Timed RTOS modeling for embedded System Design, Proc. of RTAS, IEEE (2005).
24. Hergenhan, A., Rosenstiel, W.: Static Timing Analysis of Embedded Software on Advanced Processor Architectures. Proc. of DATE, IEEE (2000).
25. Hwang, Y. , Abdi,S. & Gajski D.: Cycle-approximate Retargetable Performance Estimation at the Transaction Level, Proc. of DATE, 2008.
26. Imperas, http://www.ovpworld.org.
27. InterDesign Technologies, FastVeri (SystemC-based High-Speed Simulator) Product Overview, http://www.interdesigntech.co.jp/english/.
28. IP-XACT. The P1685 IP-XACT IP Metadata Standard. Design & Test of Computers, IEEE, Volume: 23, Issue: 4, On page(s): 316- 317, (2006).
29. ITRS - Design. International Technology Roadmap for Semiconductors. Retrieved from http://www.itrs.net/Links/2007ITRS/Home2007.htm (2007).
30. Jerraya, A. & Wolf, W. (Ed.). (2005). Multi-Processor Systems on Chip. Morgan Kaufmann.
31. Kempf, T., Karur, K., Wallentowitz, S. & Meyr, H.: A SW Performance Estimation Framework for Early SL Design using Fine-Grained Instrumentation, Prof. of DATE, IEEE (2006).
32. Klingauf, W., Gunzel, R., Bringmann, O., Parfuntseu, P. & Burton, M.: GreenBus - a generic interconnect fabric for transaction level modelling, Proc. of DAC. ACM (2006).
33. Lahiri, K. A. Raghunathan, and S. Dey: Efficient exploration of the SoC communication architecture design space. In Proc. ICCAD'00 (2000).
34. Laurent, J., Senn, E., Julien, N. & Martin, E.: Power Consumption Estimation of a C-algorithm: A New Perspective for Software Design, Proc. of LCR, ACM (2002).

35. Lyonnard, D., Yoo, S., Baghdadi, A. and A. A. Jerraya: Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip. DAC'01 (2001).
36. Madsen, J., Virk, K., and Gonzalez, M.: A SystemC abstract real-time operating system model for multi-processing sytems-on-chip. In A. Jerraya and W. Wolf. Multiprocessor Systems-on-Chip. Morgan Kaufmann (2005).
37. Magillem 4.0, http://www.magillem.org
38. Milligan, M.: The ESL Ecosystem - Ready for Deployment. Retrieved from http://www.esl-now.com/pdfs\eco_presentation.pdf (2005).
39. Moussa, I., Grellier, T. & Nguyen, G.: Exploring SW performance using SoC transaction-level modeling. Proc. of DATE. IEEE (2003).
40. Murray, B.: Virtual platforms - a reality check, part 2. SCD Source. http://www.scdsource.com/article.php?id=66 (2007).
41. NAUET Design Assembler, http://www.mataitech.com
42. Ortega, R. B and G. Borriello: Communication synthesis for distributed embedded systems. In Proc. of ICCAD'98 (1998).
43. Pasricha, S., Dutt, N. & Ben-Romdhane, M.: Fast exploration of bus-based on-chip communication architectures, Proc. of CODES/ISSS. IEEE (2004).
44. Petrot, P.: Annotation within dynamic binary translation for fast and accurate system simulation. 10th International Forum on Embedded MPSoC and Multicore (2010).
45. Popovici, K., Guerin, X., Rousseau, F., Paolucci, and A.A. Jerraya: Platform-based Software Design Flow for Heterogeneous MPSoC. ACM Transactions on Embedded Computing Systems (2008).
46. Posadas, H., Herrera, F., Sanchez, P., Villar, E., & Blasco, F: System-Level Performance Analysis in SystemC . Proc. of DATE. IEEE (2004).
47. Posadas, H., Adamez, J., Sanchez, P., Villar, E., & Blasco, P.: POSIX modeling in SystemC. Proc. of ASP-DAC'06. IEEE (2006).
48. Posadas, H., Quijano, D., Villar, E. & Martinez M.: TLM interrupt modelling for HW/SW co-simulation in SystemC. Conference on Design of Circuits and Integrated Systems, DCIS'07 (2007).
49. Posadas, H., De Miguel, G. & Villar, E.: "Automatic generation of modifiable platform models in SystemC for Automatic System Architecture Exploration". Proc. of DCIS'09 (2009).
50. Posadas, H. & Villar, E.: Automatic HW/SW interface modeling for scratch-pad & memory mapped HW components in native source-code co-simulation. In the book, Rettberg, A. et all (Eds.): Analysis, Architectures and Modelling of Embedded Systems, Springer (2009).
51. Posadas, H., Castillo, J., Quijano, D., Villar, E., Ragot, D. & Martinez M.: SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems. In the book L. Gomes & J. M. Fernandes (Eds.): Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation, IGI Global (2009).
52. Posadas, H., Villar, E., Ragot, D. & Martinez M.: Early Modeling of Linux-based RTOS Platforms in a SystemC Time-Approximate Co-Simulation Environment. ISORC, IEEE (2010).
53. Posadas, H. & Villar, E.: Modeling Separate Memory Spaces in Native Co-simulation with SystemC for Design Space Exploration. Proc. of ARCS, 2PARMA workshop (2010).
54. Qemu, http://wiki.qemu.org.
55. Schirner, G. & Domer, R.: Result Oriented Modeling, a Novel Technique for Fast and Accurate TLM, Transactions on Computer-Aided Design of Integrated Circuits, V.26, IEEE (2007).
56. SkyEye User Manual. Retreived form http://www.skyeye.org (2005).
57. Schnerr, J., Bringmann, O., Viehl , A., Rosenstiel, W.: High-Performance Timing Simulation of Embedded Software. Proc. of DAC, ACM (2008).
58. SystemC, IEEE 1666 -2005 Standard LRM, http://www.systemc.org/downloads/lrm.
59. Tanenbaum, A.: Modern Operating Systems, 2 ED: Prentice Hall (2001).
60. Viaud, E., Pecheux, F. & Greiner, A.: An Efficient TLM/T Modeling and Simulation Environment Based on Conservative Parallel Discrete Event Principles, DATE, IEEE (2006).

61. Wieferink, A., Leupers, R., Ascheid, G., H. Meyer, T. Michiels, A. Nohl and T. Kogel: Retargetable generation of TLM bus interfaces for MPSoC platforms. CODES+ISSS'05 (2005).
62. Yi, Y., Kim, D. & Ha, S.: Fast and time-accurate cosimulation with OS scheduler modeling. Design Automation of Embedded Systems, V.8, N.2-3: Springer (2003).
63. Yoo, S., Nicolescu, G., Gauthier L.G. & Jerraya, A.A.: Automatic generation of fast timed simulation models for operating systems in SoC design, Proc. of DATE, IEEE (2002).

Chapter 3
Optimization Algorithms for Design Space Exploration of Embedded Systems

Enrico Rigoni, Carlos Kavka, Alessandro Turco, Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, and Giovanni Mariani

Abstract This chapter is dedicated to the optimization algorithms developed in the MULTICUBE project and to their surrounding environment. Two software *design space exploration* (DSE) tools host the algorithms: Multicube Explorer and modeFRONTIER. The description of the proposed algorithms is the central part of the chapter. The focus will be on newly developed algorithms and on *ad-hoc* extensions of existing techniques to face with discrete and categorical design space parameters that are very common when dealing with embedded systems design. This chapter will also provide some fundamental guidelines to build a strategy for testing the performance and accuracy of such algorithms. The aim is mainly to build confidence in optimization techniques, rather than to simply compare one algorithm versus another one. The "no-free-lunch theorem for optimization" has to be taken into consideration and therefore the analysis will look forward to robustness and industrial reliability of the results.

3.1 Introduction

The optimization problems emerging in the design of embedded systems—and in particular those addressed within the MULTICUBE project— are multi-objective and characterized by the fact that all configuration parameters are discrete and possibly categorical.

Having more than one objective to be optimized (maximized or minimized) implies that the outcome of the optimization process is not a single solution but a set of solutions. This set of solutions, which is called the Pareto front, represents the trade-offs between the different (and possibly conflicting) objectives. A design is a Pareto design (or *point*) if it is not possible to improve one of its objective values without deteriorating at least another one, as in Fig. 3.1.

A. Turco (✉)
ESTECO, Trieste, Italy
e-mail: alessandro.turco@esteco.com

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_3, © Springer Science+Business Media, LLC 2011

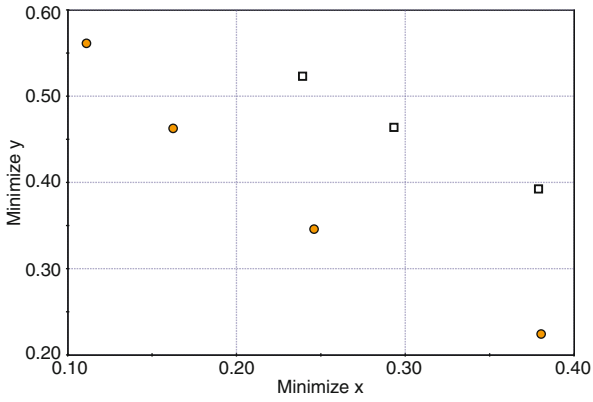

Fig. 3.1 An example of Pareto-dominance in a two-objective minimization problem. The circles represent non-dominated points, while squares are dominated points

In problems with discrete categorical configuration parameters, the types of the input variables are discrete ranges and might also be unordered collections, meaning that optimization methods which assume an order relation cannot be used profitably. For example, the number of processors on the platform is a discrete ordered variable since it is a natural number. Instead, the type of *branch predictor* to be used (e.g., static, two-level adaptive, etc.) is a categorical variable since an ordering between the different instances cannot be defined.

The evaluation of the objective values for the designs selected for exploration is usually performed through a simulator during the optimization phase. Simulators accuracy depends on their level of abstraction which is inversely proportional to their computational complexity. A manual exploration procedure might include additional delays to the already long simulation time.

Two examples of automatic optimization frameworks will be considered in this chapter. The first framework is the open source Design Space Exploration (DSE) Multicube Explorer. It has been initially conceived for the kind of problems discussed above. Throughout this chapter, a description of its optimization algorithms introduced within the framework will be provided.

The second tool is modeFRONTIER, a commercial software which has been widely used worldwide for more than ten years in different domains like aerospace, appliances, pharmaceutics, civil engineering, manufacturing, marine multi-body design, crash, structural, vibro-acoustics and turbo-machinery. All these domains define multi-objective optimization problems, but in continuous or mixed (continuous and discrete) domains, not in complete discrete and possibly categorical domains like the SoC design problems. Due to this reason, an initial re-target process to add support for categorical variables to modeFRONTIER has been carried out and the actual release of the software contains this work as well as the algorithms developed within the project.

The automatic design space exploration performed by one of these two tools is governed by an optimization algorithm. The algorithm is responsible for choosing the new configurations which have to be simulated and for analyzing the results obtained. The optimization phase can be preceded by a Design Of Experiments (DOE) study and it can be followed by some Post Processing analysis.

The optimization algorithms implement the mathematical strategies, or *heuristics*, which are designed in order to obtain a good approximation of the actual Pareto frontier. Real-world optimization problems are solved through rigorously proven converging methodologies only in an extremely few cases since the high number of input parameters and the low smoothness of objective functions limit the possible usage of classical algorithms. Therefore a wide catalogue of heuristics have been designed trying to achieve a good balance between exploration of the design space and exploitation of the information carried by the best solutions found so far.

The Design of Experiments (DOE) usually precedes the optimization stage. The aim of a DOE is to test specific configurations regardless the objectives of the optimization run but rather considering their pattern in the input parameters space. It provides an *a priori* exploration and analysis which is of primary importance when a statistical analysis has to be performed: for example, a reduced factorial DOE can be the basis for a *principal components analysis*, since it avoids correlations among input parameters and therefore it highlights input-output relationships. Moreover, almost all optimization algorithms require a starting population of designs to be considered first and the DOE can provide it, eventually generating random input values if no other preference has emerged yet.

The Post Processing analysis could represent the starting point for a new attempt of optimization, but at the same time it conveys a deep insight into the problem structure. Starting from a correlation matrix, for example, it is possible to recognize if some objectives are conflicting or if they are correlated and there is no need to involve all of them in the optimization process.

A comprehensive list of publications on this topic is out of the aim of this introduction. The description of the algorithms listed in Sect. 3.3 will include the references necessary to understand them. The theoretical structure of multi-objective optimization as well as classical methods are deeply investigated in the book by Miettinen [10]. The paper by Erbas et al. [5] describes in details a MPSoC design problem solved with genetic algorithms (GA) and it introduces important concepts like evaluation metrics and repair mechanisms.

The innovation of the algorithms developed within the MULTICUBE project cannot be correctly evaluated without considering the whole picture: it is mathematically proven that it is not possible to rank optimization algorithms on the basis of their performance over all possible problems. On the contrary, it is possible to specialize an optimization strategy in order to solve "better" (later in this chapter the concept of quality for a multi-objective solution set will be addressed more precisely) a defined class of problems. The work done by MULTICUBE partners generated in a very satisfactory trade-off between applicability and accuracy which is the true achievement of the project. Not only algorithms contributed to this result and this is the reason why this chapter also introduces some features of the software framework containing them and an anticipation of the validation procedure necessary for an industrial knowledge transfer.

This chapter is organized as follows: Sect. 4.2 presents the problem description and the software framework while Sect. 4.3 introduces the design space exploration algorithms used throughout the project. Section 4.4 presents a detailed descriptions of validation strategy while Sect. 4.5 summarizes the main content of this chapter.

3.2 Problem Description and Software Framework

To define an optimization problem it is necessary to identify the input parameters of the problem, which constitute the Design Space, and the output parameters, often called *metrics*. Among the output parameters some objectives must be selected in order to build the Objective Space. Other outputs can be considered as constraints, but also input parameters can be combined in order to obtain the desired constraints. A typical example is the pair of variables processor type-cache size: some processors may support only certain values of cache size and a constraint must be added to the problem to enforce this requirement.

The output values associated to a given configuration of input parameters are obtained through a simulator. The choice of which designs have to be explored is responsibility of the optimization algorithm and of the DOE designer for what concerns the starting points. The whole structure comprising inputs, outputs, simulator, algorithm is called workflow.

One of the first achievements of the MULTICUBE project is the definition of a common framework for generating a complete workflow using a standard XML format. Both DSE tools, Multicube Explorer and modeFRONTIER, can accept the same configuration file and they are able to run optimization starting from the information listed in it. Specific tags have been created in order to specify input and output values, a precise syntax is used to define constraints and the path to the executable simulator is included.

The communication between the optimization algorithm and the simulator is performed through XML file as well. The DSE tool is also responsible for launching parallel instances of the simulator, if the computational resources available allow them.

From the strictly mathematical point of view, the problems addressed can be characterized as follows. The vector of input variables $\mathbf{x} = (x_1, \ldots, x_N)$ can take values into different sets depending on the nature of its components. An integer variable is usually comprised between a lower and an upper bound, $x_i^L \leq x_i \leq x_i^U$, where $i \in [1, N]$. However it is possible that only some integer values might be of interest, for example only the powers of 2. This kind of variables are similar to categorical variables, but they maintain the notion of order. On the contrary, another way of calling a categorical variable is *discrete unordered* variable since this is their main feature. The list of admissible values is called *catalogue*. The optimization problem is then

$$\begin{cases} \min & \mathbf{f}(x_1, \ldots, x_N), \\ \text{such that} & \mathbf{g}(x_1, \ldots, x_N) \geq 0, \\ & \mathbf{h}(x_1, \ldots, x_N) = 0. \end{cases} \tag{3.1}$$

The vector functions $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{h}$ can have arbitrary dimensions. It is not necessary that all the functions involved in the problem are minimization targets. For example if a function $\phi(\mathbf{x})$ has to be maximized and it should occupy the i-th component of $\mathbf{f}$, a simple change of sign can solve the problem defining $\mathbf{f}_i := -\phi$. The same procedure can be applied to "less or equal to" constraints.

A point is said to be *feasible*, if it satisfies all the constraints addressed in $\mathbf{g}$ and $\mathbf{h}$. If the number of objectives is M, then given two feasible points $\mathbf{x}$ and $\mathbf{y}$, the point $\mathbf{x}$ is said to *dominate* $\mathbf{y}$ if $\mathbf{f}_i(\mathbf{x}) \leq \mathbf{f}_i(\mathbf{y})$ for $i = 1, \ldots, M$ and there exists at least one index $j \in [1, M]$ such that $\mathbf{f}_i(\mathbf{x}) < \mathbf{f}_i(\mathbf{y})$. Dominance induces a partial ordering onto the design and the objective space.

A vector of input parameters $\bar{\mathbf{x}}$ is said to be a *Pareto design* if there is not any other point dominating it. The corresponding point in the objective space $\mathbf{f}(\bar{\mathbf{x}})$ is said to be a *Pareto point*. The set containing all the Pareto points is the *Pareto front* and it is the solution of problem 3.1.

3.3 Algorithms

This section presents a brief description of the multi-objective optimization algorithms that have been tested within the project and implemented in the Design Space Exploration tools, highlighting the algorithm characteristics that are related to the specific properties of the optimization of an embedded system. The algorithms presented in this section can be divided in three groups:

- Standard algorithms. This first group includes the algorithms that are well known in the multi-objective optimization field and have been implemented in the Design Space Exploration tools by following the models proposed by their original designers with none or minimal enhancements. The algorithms NSGA–II and MOGA–II belong to this group.
- Enhanced algorithms. This group includes all algorithms that are based on a previously defined algorithm but include noticeable enhancements that make them adequate for the specific problems addressed. The algorithms Enhanced-MOSA, Enhanced-ES and Enhanced-MOPSO belong to this group.
- New algorithms. This group includes all algorithms that have been specifically defined in the MULTICUBE project for multi-objective optimization in the context of System-on-Chip (SoC) design optimization. The algorithms MFGA and APRS belong to this group.

3.3.1 Standard Algorithms

The algorithms described in this section have been employed successfully in multi-objective optimization for years. They were not precisely designed for categorical variables, but they can treat them as simply discrete ones without excessively deteriorating their performances.

3.3.1.1 NSGA–II

The Non-dominated Sorting Genetic Algorithm II (NSGA–II) was developed by Prof. Deb et al. [3] at Kanpur Genetic Algorithms Laboratory (KanGAL). NSGA–II

is a fast and elitist multi-objective evolutionary algorithm which shares the basic structure with all genetic population-based algorithms.

The basic mechanism can be summarized as follows: starting from a parent population, some individuals are selected for generating a child population. The algorithm applies operators that work on the input variables of the selected individuals trying to improve their outputs: mutation and crossover are the standard choices for NSGA–II. This procedure is iterated for the requested number of generations.

This algorithm has some peculiar and powerful characteristics:

- It includes a fast non-dominated sorting procedure. Sorting the individuals of a given population according to the level of non-domination is a complex task, which makes in general non-dominated sorting algorithms computationally expensive for large population sizes. The adopted solution in NSGA–II performs a clever sorting strategy.
- The multi-objective search includes elitism. NSGA–II implements the multi-objective search using an elitism-preserving approach, which is introduced storing all non-dominated solutions discovered so far, beginning from the initial population. Elitism enhances the convergence properties towards the true Pareto-optimal set.
- A parameter-less diversity preservation mechanism is adopted. Diversity and spread of solutions is guaranteed without the use of extra parameters (like sharing parameters for example). NSGA–II adopts a suitable parameter-less niching approach called crowding distance, which estimates the density of solutions in the objective space, and a crowded comparison operator, which guides the selection process towards a uniformly spread Pareto frontier.
- The constraint handling method does not make use of penalty parameters. The algorithm implements a modified definition of dominance in order to solve constrained multi-objective problems efficiently: usual dominance is the criterion to sort feasible points. A feasible point will be always preferred to an unfeasible one. Unfeasible points are sorted looking at the (normalized, possibly) constraint violations sum.

The NSGA–II procedure is described graphically in Fig. 3.2. The individuals of the parent population P_t of size N and the new population Q_t of the same size (created by applying the variation operators crossover and mutation, to individuals in P_t selected by binary tournament) are grouped together. The combined population R_t is then sorted based on its non-domination level obtaining sets of non-dominated solutions (F_1, F_2, ...). The new population P_{t+1} is created by selecting the best non-dominated sets that can be completely inserted into the new population (with a combined size smaller or equal than N) plus members from the last set (which cannot be fully accommodated) selected using the crowded comparison operator.

NSGA–II allows both continuous (real-coded) and discrete (binary-coded) design variables. Specific mutation and crossover operators are applied to each kind of variables. Categorical (non-ordered discrete) parameters are treated as simple discrete ones. This drawback can be compensated by increasing the exploration capabilities of the algorithm allowing a larger mutation probability. NSGA–II tests a

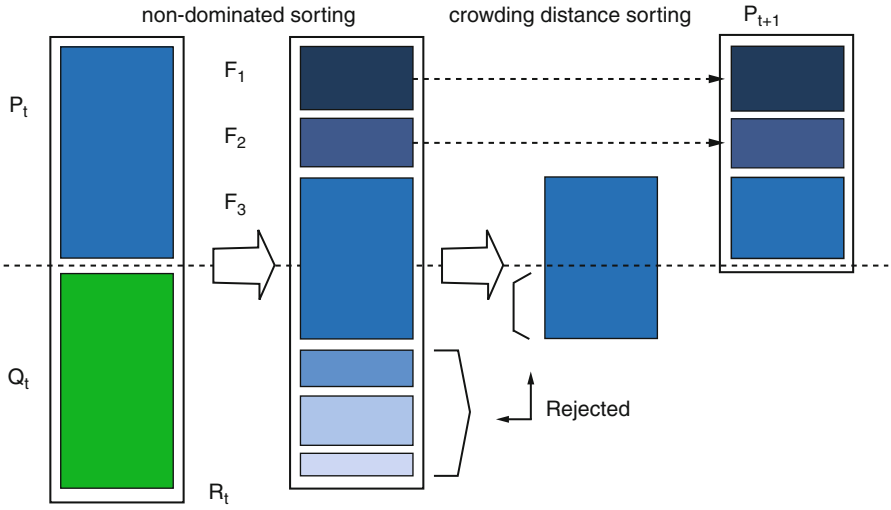

Fig. 3.2 NSGA–II sorting procedure

wider range of candidate solutions (hence the fictitious locality is damped) and its efficient elitism and selection routines drive the optimization towards the Pareto front. This algorithm has been implemented in Multicube Explorer while it was provided by modeFRONTIER from early releases.

3.3.1.2 MOGA–II

MOGA–II (Multi-Objective Genetic Algorithm, with elitism) was originally formulated by Poloni [11] and it reached the actual implementation with the introduction of elitism. This second version is equipped also with an optional improved crossover operator (directional crossover), but since it is not suited for discrete problems, its description is omitted.

This algorithm accepts only discrete variables (possible continuous variables have to be discretized with the desired accuracy), which are encoded as in classical genetic algorithms. Three operators govern the reproduction phase: one-point crossover, mutation and selection. The probabilities under which one of them is chosen are user-defined parameters.

Elitism guarantees that the best points remain in the parent population and hence hopefully their children will exhibit a similar behavior. MOGA–II achieves this result keeping a record of all non-dominated points found up to the current generation. The new population is created extracting randomly the requested number of new parents from the union (without repetitions) of the elite set and the set of newly generated children. This procedure gives to the algorithm a good balance between exploration and exploitation phase. This balance is fundamental for the performance of a multi-objective global search: the search space has to be sufficiently explored, but at the

same time the number of evaluated points must be kept low and the algorithm has to converge rapidly to the Pareto set.

Elitism in MOGA–II works in this direction. The elite set usually contains a few points in early stages of the optimization. Moreover, these points belong to the last generation with a high probability. Hence only a fraction of them will enter the next parents population promoting exploration. As long as new generations are created, the elite set grows and the probability of finding out a new elite point decreases. Therefore in the updated population there will be many points coming from the elite set exploiting their features.

In order to save simulation time, steady evolution was preferred against the classical generational evolution in MOGA–II. In almost any industrial application, the computational time spent in evaluating a point is much larger than the time employed by the optimization algorithm to prepare and request a new evaluation. A generational algorithm would keep a significant part of computational resources idle (in a cluster or grid systems), since before every generation is created, the algorithm needs to get the results of the evaluation of all the individuals of the previous generation.

Within a steady evolution, the child point replaces its parent immediately. Every time an evaluation ends, a new one is requested choosing randomly a parent from the actual population and applying the chosen operators. The elitism procedure is scheduled with the same frequency as in the case of a generational evolution, but it can be performed while some points are still being evaluated, using the information stored so far. This introduces a little delay in the propagation of the information, but it prevents delays in the computational grid or cluster system. The negative effects of this issue increase with the dimension of the population, but decrease as the number of requested generations increases. This algorithm is proprietary of ESTECO and it was provided by modeFRONTIER from early releases.

3.3.2 Enhanced Algorithms

Literature reports a continuous improvement of available methods since multi-objective optimization is a research field constantly pressed by new applications. New applications require new and better answers. In this section we considered as *enhanced* algorithms the implementations of well known algorithms which have been rewritten within the MULTICUBE project in order to better adapt to the SoC design problem. Indeed, specific operators have been designed for treating categorical variables and a careful attention has been addressed to the problem of optimizing also the computing resources needed for design evaluation.

3.3.2.1 Enhanced-MOSA

The Simulated Annealing (SA) method for optimization was introduced by Kirkpatrick [6], on the basis of a thermo-dynamical analogy.

The evolution of such a system is controlled by an external parameter called temperature. A related energy can be assigned to every possible configuration of the

system. When an initial configuration is perturbed, the difference in energy between the two states is responsible for the evolution of the system: if the new state is favorable, i.e. if it decreases the energy, then the new configuration is accepted. If this is not the case, the new state is accepted or rejected according to a probability distribution derived by Boltzmann. This distribution is a function of the temperature and when the temperature is high, the probability of accepting an unfavorable state is larger (see Metropolis et al. [9]).

The energy for the MOSA algorithm is (a suitable function of) the non-dominated ranking already described for NSGA–II. Hence a new point is always accepted if it is non-dominated by its parent. It will be also accepted in the opposite situation depending on a temperature-based probability distribution.

Temperature is simply a parameter, initially user-defined, which evolves during the optimization. MOSA starts with a hot phase accepting many points in order to explore the design space. Afterwards, a cold phase, during which only the best points survive, represents the exploitation part of the algorithm.

The creation of a child configuration from a parent one in the original formulation of the algorithm is a directional perturbation. A random direction *versor* represents the direction of the perturbation, while its length is predicted by a schedule similar to the temperature one: starting from a specified upper bound, the value decreases during the hot phase and it reaches the imposed lower bound in the cold phase. If the perturbation vector brings the point out of variables space boundaries, a bouncing routine will maintain the feasibility of the samples. This procedure also helps in differentiating and enhancing the exploration and the exploitation capabilities of the algorithm.

The concept of direction has no meaning working with categorical variables. The enhanced version of MOSA takes in account this problem keeping at the same time the idea of a tunable perturbation. Every categorical variable at each iteration has a probability to change its value depending on a law similar to the one applied to the temperature and perturbation length. If the value has to be changed, a new value is chosen randomly from the available list.

A lifespan counter is introduced in order to compensate for the uncontrolled randomness in the search for the best values for categorical variables. Especially during the hot phase there could be sequences of parents and children moving towards dominated regions of the objective space because of the Metropolis acceptance criterion. If the number of subsequent unwanted increasing in energy exceeds a threshold, Enhanced-MOSA replaces the child with its better-fitting parent.

A steady state evolution is a second enhancement of the algorithm. The procedure is very similar to the evolution implemented in MOGA–II with the obvious change of the updating schedule: there is no elite set to be updated, but instead Enhanced-MOSA changes the value of the temperature, the perturbation length and the probability of replacement for categorical variables.

The standard implementation of MOSA is available in modeFRONTIER from early releases and it has also been implemented in Multicube Explorer. The described enhancements were developed for the MULTICUBE project and were implemented in modeFRONTIER.

3.3.2.2 Enhanced-ES

Evolution Strategies (ES) is a multi-objective optimizer that is able to reproduce different evolutionary algorithms. These algorithms share the selection operator and the operators schedule, while they differ in the ratio between parents and children points and in the definition of the set of points among which the new parents are selected.

The ES approach was first used at the Technical University of Berlin. During the search for the optimal shapes of bodies in a flow, the classical attempts with the coordinate and the well-known gradient-based strategies were unsuccessful. So, the idea was conceived of proceeding strategically. Rechenberg and Schwefel [14] proposed the idea of trying random changes in the parameters defining the shape, following the example of natural mutations.

Usually, there is a huge difference between mathematical optimization and optimization in the real-world applications. Thus, ES were invented to solve technical optimization problems where no analytical objective functions are usually available.

The general Evolutionary Strategy scheme is the following:

1. Initial population creation;
2. Individuals evaluation;
3. Selection of the best individual(s);
4. Recombination;
5. Mutation;
6. Individuals evaluation;
7. Return to step 3 until the required number of generation is achieved

Selection of the best results may be done only on the set of children or on the combined set of parents and children. The first option, which is represented usually with the notation (λ, μ)-ES, can "forget" some good results when all the children are worse than their parents. The second option, which is represented by the notation $(\lambda + \mu)$-ES, applies a kind of elitist strategy for the selection.

The best solutions may be identified in different ways: the implementation of ES provided in modeFRONTIER is capable of approximating the Pareto Set in multi-objective optimization by using the Non-dominated/Crowding distance sorting technique as done in NSGA–II.

The main source of variation is a mutation operator based on a normal distribution. The standard deviation of this distribution changes during the generations in an adaptive manner. Each input variable has its own deviation with an initial and a minimal value that can be arbitrarily tuned.

A completely different operator has been introduced for categorical variables in the context of MULTICUBE. If such a variable is selected for mutation, its value is changed by following an uniform distribution (i.e. the choice is completely random), since locality has no meaning. An adaptive strategy is performed over the probability of mutating each variable.

A discrete recombination operator is a second source of variability. It resembles the classical crossover operator, where information coming from two different parents

is exchanged producing a child. The value of each variable has the same probability of coming from both parents.

The standard implementation of ES is available in modeFRONTIER from early releases and it has also been implemented in Multicube Explorer. The described enhancements were developed for the MULTICUBE project and were implemented in modeFRONTIER.

3.3.2.3 Enhanced-MOPSO

Particle Swarm Optimization (PSO) is an optimization methodology that mimics the movements of a flock of birds finding food [7]. PSO is based on a population of particles moving through an hyper-dimensional search space. Each particle possesses a *position* and a *direction*; both variables are changed to emulate a well known social-psychological phenomenon: mimic the success of other individuals in the population (also called *swarm*).

More formally, the position $\mathbf{x}$ for a single particle i is updated by means of a velocity vector *vecv* by means of the following equation:

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \delta_i(t) \tag{3.2}$$

while the direction vector is updated with the following equation:

$$\begin{aligned}\delta_i(t) = W\delta_i(t-1) + C_1 r_1(\mathbf{x}_{pbest_i} - \mathbf{x}_i(t-1)) \\ + C_2 r_2(\mathbf{x}_{gbest} - \mathbf{x}_i(t-1))\end{aligned}$$

where W is called the inertia weight, C_1 is the cognitive learning factor, C_2 is the social learning factor, r_1, r_2 are random numbers in the range $[0, 1]$, $\mathbf{x}_{pbest_i}$ is the best position of particle i with respect to the minimization problem, $\mathbf{x}_{gbest}$ is the global best found up to time t. The formulation of the problem leads to solutions which try to 'follow' the leader's $\mathbf{x}_{gbest}$ position as well as attracting solutions versus the *personal* best solution of the particle $\mathbf{x}_{pbest_i}$.

Dealing with Multi-objective problems. So far, several approaches have been proposed for extending the formulation of the PSO technique to the multi-objective domain [2, 13]. The Enhanced-MOPSO technique is based on an "aggregating" approach where the swarm is equally partitioned in n subswarms, each of which uses a different cost-function which is the product of the objectives combined with a set of exponents randomly chosen.

In other words, given the original set of objectives $\{f_1 \dots f_m\}$, each sub-swarm i solves the following problem:

$$\min_{\mathbf{x}\in X} \prod_{j=1\dots m} f_j^{p_{i,j}}(\mathbf{x}) \tag{3.3}$$

where $p_{i,j}$ is a set of randomly chosen exponents. It can be shown that solutions to Problem 3.3 lie on the Pareto surface of the original problem. This approach

is different with respect to [2] because the latter uses a linear combination of cost functions $\{f_1 \dots f_m\}$. Linear combination can be heavily biased on highly valued cost-functions disregarding low-valued ones.

Dealing with the discrete design space. The essential nature of the solution space of the problems faced with the MULTICUBE project is discrete, while the PSO approaches presented so far deal with a continuous search space. Several proposals have been made so far in the literature to extend classical PSO to the discrete domain. One method for addressing the discrete design space exploration problem is applying particle swarm optimization to binary-valued solution elements [8]. In this case, while the velocity term is still real-valued, the position term is actually chosen between 0 and 1 by means of a sigmoidal function. Another method leverages the construction of a probability distribution for each value of the position vector [12]. The probability distribution is derived from the current value of the position vector which, in turn, depends on the velocity vector. The probability distribution is then transformed into a single integer number during fitness evaluation of each particle.

Enhanced-MOPSO is based on the concepts of *random walk* theory. A random walk is a path with the following properties:

- It has a starting point.
- The distance from one point to the next is constant
- The direction from one point to the next is picked up at random.

The position of the particle is still updated with the traditional rule:

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \delta_i(t) \tag{3.4}$$

while each component k of the direction vector is updated with the following rule:

$$\delta_{i,k}(t) = \begin{cases} \text{sign}(x_{gbest,k} - x_{i,k}(t-1)) & \text{if(rand()} < p) \\ \text{randint}(-1, 1) & \text{otherwise} \end{cases} \tag{3.5}$$

where $p \in [0, 1]$ is a parameter of the algorithm.

As can be noted, the direction of the particle is updated following two rules: rule 1 attracts the particle versus the leader of the swarm ($gbest$), rule 2 forces the particle to follow a random walk. This ensures us to jump out from local minima in the objective function shown in Eq. 3.3.

3.3.3 New Algorithms

The algorithms listed below can be considered as completely new proposal in the scientific literature [17].

3.3.3.1 MFGA

The acronym of this new algorithm stands for Magnifying Front Genetic Algorithm since its main purpose is to work on the local Pareto front in three directions: towards (approaching the true front), laterally (obtaining a wider front) and internally (enhancing the uniformity of the front samples).

With the introduction of elitism, genetic algorithms such as NSGA–II found a very good answer to the problem of converging faster than previous implementations. The question now is how to converge better, without slowing down.

Elitism is considered as the main reason of too concentrated Pareto fronts [1, 4]. If the optimization problem is difficult, only a few points will be non-dominated and will become a sort of basin of attraction. Indeed, elitist strategies will keep these points in the parent population and crowding distance or similar techniques are not useful to "dilute" them until a large number of Pareto points are found. However without elitism the request for quality cannot even be addressed, since the algorithm would converge too slowly. Literature reports two promising ideas in order to modify this operator without removing it.

Deb and Goel [4] proposed a controlled elitism approach. Their algorithm selects the new parent population accepting also dominated points with a preference for those points coming from less crowded regions. Computed points are ranked by domination and ordered by crowding distance. An exponentially decreasing number of points are selected from each rank starting from the top of the list. Figure 3.3 shows how a combined population R_t of size $2N$ (parents plus children) is reduced

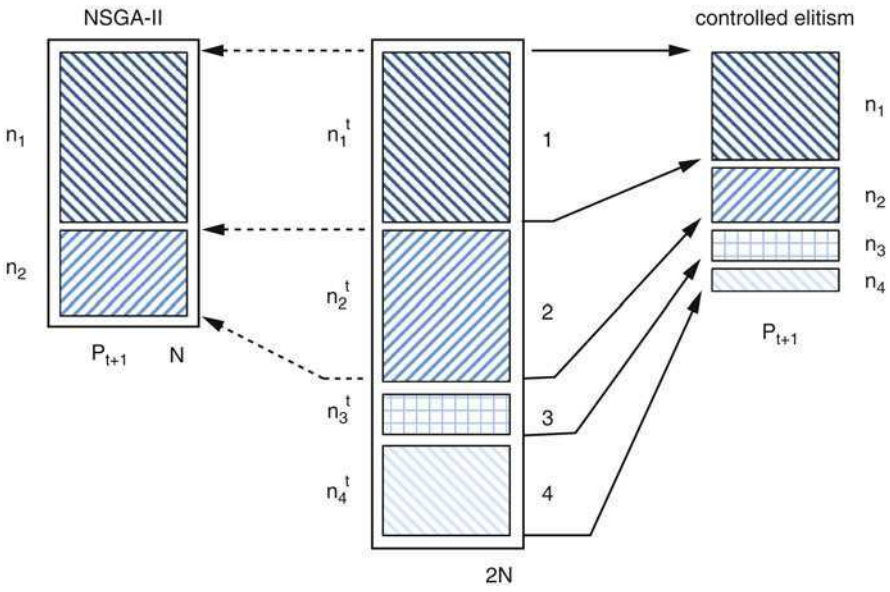

Fig. 3.3 Controlled Elitism sorting procedure vs NSGA–II

to a population P_{t+1} of size N by selecting n_i individuals from each non-dominated front of size n_i^t, using crowded tournament selection to reduce each front if necessary.

This choice helps in obtaining a more uniform front, filling possible gaps with points coming from higher ranks. It is remarkable that slight improvements are achieved also in convergence rate and in lateral spreading of the computed front, as reported in the cited paper.

Aittokoski and Miettinen [1] studied a different strategy called variable size population. Their idea is to transform all first-rank points into new parents regardless their number. The result is an algorithm that cannot perform worse than a classical elitist one considering convergence rate (since it does not waste any useful information) and it guarantee a better diversity maintenance.

MFGA is an algorithm that tries to *manage elitism* mixing the two cited ideas in an original scheme, including also a steady state evolution. The algorithm switches automatically between the two approaches depending on the dimension of the local Pareto front, allowing:

- wider exploration of the design space: new generations are created by following the reduced elitism approach until the local Pareto front reaches one third of the population size (fixed by the DOE size).
- better exploitation of the obtained information: the parents update is done by following the variable size scheme, only for local fronts that contain a number of points from one to two third of the population size.
- diversity preservation: the reduced elitism is reintroduced for larger fronts.

The steady state evolution implemented together with this mixed procedure is quite demanding from the computational point of view, since every time the evaluation of a new point is performed, the parent population is completely recomputed including the new achieved information. This choice is well suited for problems involving long simulation time.

Classical operators govern the parents-children recombination, but they are rebuilt trying to enlarge the kind of problems treatable using them [17]. Mutation and crossover operators act in MFGA variable-wise in order to treat easily mixed problems involving real, integer and categorical variables. MFGA was completely designed by ESTECO for the MULTICUBE project. Its inclusion in future commercial releases of modeFRONTIER is planned.

3.3.3.2 APRS

The acronym of this new algorithm stands for Adaptive-windows Pareto Random Search. It is an iterative optimization algorithm that tries to optimize locally each Pareto solution found up to the previous iteration.

The main characteristics of the algorithm are represented by the three keywords: *adaptive-windows*, *Pareto* and *random search*.

- *Adaptive-windows*: the APRS is an algorithm that has dynamic windows size which is reduced with the time spent in the exploration and with the goodness of the point found in current windows.

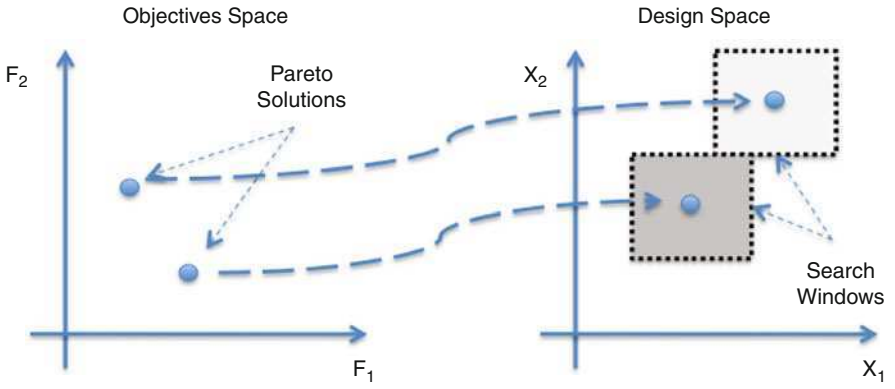

Fig. 3.4 Search windows idea in the APRS algorithm

- *Pareto*: the starting points of each iteration of the algorithm are the current Pareto solutions (found up to previous iteration) where the search windows are centered on.
- *Random search*: the new configurations to be evaluated are randomly selected within those windows.

The idea behind the algorithm is simple and it is based on iterative local search in the neighborhood of good points: the current Pareto set (see Fig. 3.4). Moreover, as for temperature in Simulated-Annealing strategies, the algorithm has a parameter represented by the search-window size that reduces the dynamism of the algorithm during the optimization life-time. In particular, the search-window size centered on each Pareto point is reduced whenever no better solutions are found in the iteration.

More in detail, the general structure of the algorithm is the following:

1. Creation of an initial set of solutions selected within the entire DS by using a DoE
2. Initial solutions evaluation (S)
3. Identification of the Pareto solutions ($P' = ParetoFilter(S)$)
4. While the termination condition is not met
   (a) Use the Pareto set as initial set for each iteration $S < -P'$
   (b) For each point in the Pareto set ($p \in P'$)
      i) Randomly select a configuration within the reduced DS delimited by the search window centered on the pareto point ($W(p)$)
      ii) Evaluate the selected configuration s ($S = S + s$)
   (c) If the Pareto set is not changed from the previous iteration (i.e. if $P' == ParetoFilter(S)$), reduce the search window ($|W| = |W| * \alpha$)
   (d) Otherwise, Identification of the new Pareto set ($P' = ParetoFilter(S)$)

The algorithm exposes three parameters, the initial set $|S|$ size, the initial size of the windows $|W|$ and the windows reduction coefficient α. APRS was completely designed by POLIMI for the MULTICUBE project and has been implemented in Multicube Explorer.

3.4 Validation Strategies

The so called *no-free-lunch theorem for optimization* [18] is a rigorous mathematical theorem which states the impossibility of ranking algorithms on the basis of their performance: averaging on all possible problems, every algorithm will obtain results exactly equal to all the others. A direct corollary is that if one algorithm appears better than another one in solving a specific problem, there must exist another problem in which the original algorithm appears worse than the other.

This theorem implies that an optimization algorithm is as important as the validation strategy which may reveal its ability in solving a specific set of problems. This issue is not a mere academic question, but it has relevant effects on the industrial exploitation of the achievement obtained. It is of primarily importance to build confidence on the proposed algorithmic strategies and to prove their robustness. This is the main achievement of the researches carried on by the MULTICUBE project, whose partners agreed on the need of a validation step in order to transfer the (possibly academic) high quality knowledge to the industrial world.

This section focuses on the first two steps performed in this direction within the project. The first one consists in showing that all the algorithm described in Sect. 3.3 can solve a benchmark problem of SoC design in a satisfactory manner. The second step is to show the advantages of an automatic optimization process with respect to a traditional approach. The combination of these two results can guarantee the reliability of the proposed approach. At the same time, the validation process must be intended as an iterative process which follows but at the same time precede the development of new optimization strategies.

3.4.1 Algorithm Comparison

The problem selected as benchmark for the algorithms validation is based on the SP2 low-power processor use case delivered by STM-China described in Chap. 8. In this paragraph, we compare the previously introduced algorithms to identify the most suitable for the architecture under consideration. The executable model for the design space exploration is the *sp2sim* simulator, which models the SP2 microprocessor design. The benchmark application selected is the *164.gzip* application, based on the popular gzip application.

The design space consists of 11 configuration parameters, 7 system metrics and 3 objectives. The configuration parameters are grouped in three categories: out-of-order execution parameters, cache system parameters and branch prediction parameters, as shown in Table 3.1. The system metrics are grouped in three categories: performance, power dissipation and area occupation metrics, as shown in Table 3.2. The three objectives to be minimized have been selected from each one of the metrics group: total_cycle, power_dissipation and area.

In order to compute optimization metrics that provide a reasonable measure of quality of the algorithms, it is necessary to compare the Pareto fronts obtained by

Table 3.1 Input parameters for the benchmark problem

Category	Parameter	Description	Values
Out of order execution	rob_depth	Reorder buffer depth	32, 48, 64, 80, 96, 112, 128
	mreg_cnt	Rename register number	16, 32, 48, 64
	iw_width	Instruction window width	4, 8, 16, 24, 32
Cache system	icache_size	Instruction cache size	16, 32, 64
	dcache_size	Data cache size	16, 32, 64
	scache_size	Secondary cache size	0, 256, 512, 1024
	lq_size	Load queue size	16, 24, 32
	sq_size	Store queue size	16, 24, 32
	mshr_size	Miss holding register size	4, 8
Branch prediction	bht_size	Branch history table size	512, 1024, 2048, 4096
	btb_size	Branch target buffer size	16, 32, 64, 128

Table 3.2 Output parameters for the benchmark problem

Category	Metric	Description
Performance	total_cycle	Total cycle number
	total_instr	Total instruction number
	IPC	Instruction per cycle
Power dissipation	total_energy	Total energy consumed
	total_dissipation	Average power dissipation
	peak_power_dissipation	Peak power dissipation
Area occupation	Area	Area occupied

the algorithms with a reference Pareto front, which should be the real Pareto front of the optimization problem, or at least a good approximation of it. The design space of the problem outlined above consists of 1,161,216 designs. Since the time required to evaluate all these designs is too large to be considered as an option, a statistical study of the configuration parameters was performed in order to try to identify parameters which could be fixed to a constant value to reduce the size of the design space without a significant reduction of the problem interest for SoC designers. A statistical study was performed with random exploration using Multicube Explorer, exploring a set of 5,000 randomly selected designs.

In order to reduce the size of the design space, the following parameters with low contribution where identified: rob depth, lq size, sq size and mshr size. Adequate constant values were selected for them, reducing the size of the design space to only 9,216 designs. The reduced problem was considered valid both from the point of view of SoC designers and from the point of view of the mathematical properties of the design space and its associated Pareto front. All designs in the reduced design space were evaluated by performing a full factorial multi-level exploration obtaining the real Pareto front in a few days of execution time. This Pareto front consists of 18 points. Figure 3.5 shows if and when the considered algorithms discover them.

The performance measures selected for the algorithms comparison concern both time and quality. Since by far the most time consuming component of the optimization procedure is the simulator execution, its number of evaluations has been selected as a fair measure of the required algorithm execution time. Concerning the quality of the

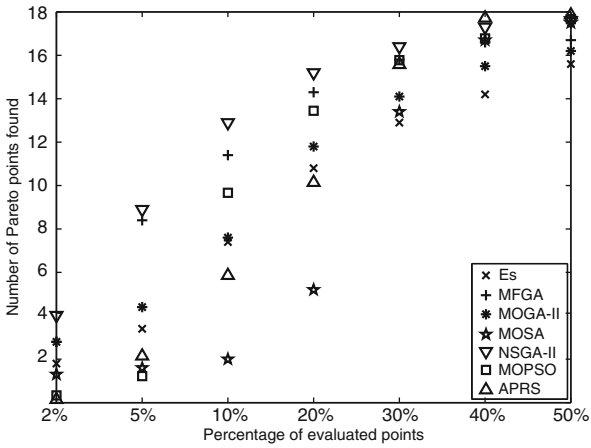

Fig. 3.5 Algorithms performance comparison on the reduced benchmark problem

solutions found by each algorithm, a set of four metrics has been selected considering the following criteria:

- Accuracy: measured as the distance between the obtained Pareto front and the reference (or real) front.
- Uniformity: measured as the distribution of the solution set in the trade-off curve.
- Extent: measured as the coverage of the objective space considering boundary points.

The D-metric, Δ-metric and ∇-metric have been selected from [5], while the ADRS (Average Distance from Reference Set) metric has been selected from [15]. Both D-metric and ADRS provide indication of accuracy, Δ-metric of uniformity and ∇-metric of extent.

A fair evaluation of non-deterministic algorithms requires several repeated runs without changing any parameter besides the random generator seed. Notwithstanding the relative small search space consisting of only 9,216 designs, very large variations can be observed in the algorithms behavior and a rigorous study needs to analyze also this aspect. It was proved that 10 repetitions were a good trade off among statistical issues, purposes of the evaluation and significance of the problem. Preliminary tests were performed in order to estimate the best choices for the tunable parameters which then have been kept fixed.

Algorithms parameters are usually problem-dependent. Some of them depend also on the user expectations: the optimal choices (if any) for parameters controlling the ratio between exploration and exploitation phase (like temperature schedule in MOSA, for example) are strictly related to the number of evaluations the user can afford. It was decided to tune these parameters considering the largest target (i.e. 50% of the design space, as described below) and accepting possible worse results in the partial samples.

The evaluation process then proceeds checking at fixed numbers of explored points the quality of the non-dominated front found so far. The steps selected for the

evaluation are: 184 designs (corresponding to about 2% of the design space), 461 (5%), 922 (10%), 1,843 (20%), 2,765 (30%), 3,686 (40%) and 4,608 (50%). Only the requests of evaluation of new designs were counted, since sometimes the algorithms request the evaluation of an already evaluated design due to the inherent behavior of their random engines. In any practical application, the time needed to retrieve the stored information is incomparably smaller than the time that would be spent by a new evaluation. Besides the fact that in this experiment it is known in advance any value thanks to the previous full factorial exploration, the real optimization process was simulated by counting each design only once.

Some algorithms occasionally cannot generate new designs when working with discrete problems if some parameters are not set properly. The chosen benchmark problem has a small variable space and in the exploitation phase (where usually recombination is less effective than in the exploration one) the algorithms may get stuck in the endless repeated evaluation of the same designs. This behavior was observed and was overcame by increasing the exploration capabilities of the algorithms.

A last remark concerns the input variables nature. They are all discrete, but none of them is categorical. This choice allows to test fairly a wider range of algorithms, but on the other hand, the test cannot highlight completely the improvements gained with the enhancements described above.

With a small variance, all algorithms reach an ADRS value below 2% evaluating 30% of the design space (see Fig. 3.6). This result can be considered very promising. Variations in the slope of the lines for some algorithms are a consequence of possible different behaviors in successive phases of the optimization process. The most clear example is MOSA with its hot and cold phase. MOSA is tuned to reach the top of the exploitation phase at 50% of evaluations and therefore its results are the worst up to 20–30%, while at the end it is one of the most effective algorithms. APRS shows a similar behavior.

It is very difficult to analyze the uniformity and the extent of the partial front found by the algorithms during the optimization process. The true Pareto front is

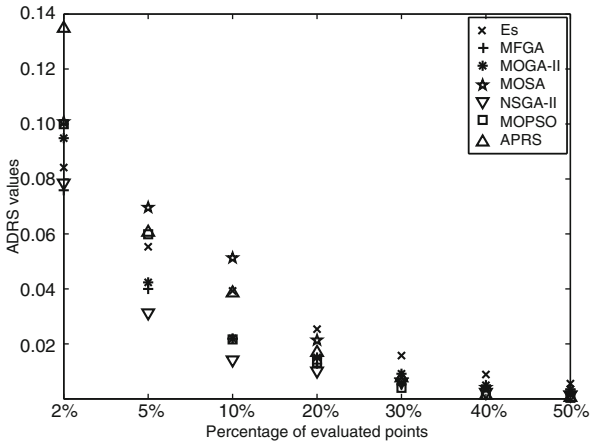

Fig. 3.6 Algorithms performance comparison on the reduced benchmark problem in terms of ADRS metric [15]

not uniform itself, since the problem is highly discrete both in the search space and in the objective space: notwithstanding the real values achievable by the objective functions, it is observed the formation of clusters of points with a cylindrical shape. Only some tips belong to the Pareto front and the distance between two nearby solution points is relatively large.

Extent metric gives some insight into the evolution of the non-dominated front. Some algorithms (MOPSO, MOSA, and APRS) span a wider range than others. This is due to a sharper division between exploration and exploitation phase. Indeed, the higher values of ∇-metric are achieved when the local front contains points which will be dominated by the following generations. These designs may span a wider area in the objective space resulting in a higher value of the extent metric.

The analysis of the metrics values obtained offers a deep insight into the algorithms structure in addition to the comparison information. Efficiency assessments can be drawn in terms of ADRS metric and number of Pareto points found. Under this perspective, all algorithms behave in a satisfactory manner on the proposed benchmark problem: as remarked before, starting from 30% of the design space exploration, all scored less than 0.02 in ADRS metric. The worst score in the achievement of Pareto points can be taken as an indicator of the reliability of the proposed algorithms: ES found 15.6 points which however corresponds to 86.6% coverage of the true Pareto front. Since in real-world problems the solution set is unknown, this percentage is clearly a good guarantee that the algorithms will reach at least a significant part of the Pareto front.

3.4.2 The Complete Optimization Problem

The problem proposed for algorithms comparison can be very useful for validating the whole optimization process as well. The procedure followed for obtaining the subspace of points for the comparison can be summarized as:

- simulate an initial set of 5,000 random points;
- perform statistical analysis on the sample in order to detect the most significant variables and the most appropriate values for the remaining ones;
- simulate all the configurations (full factorial exploration, 9,216 points) obtained varying the selected variables;
- extracting the non-dominated set.

The approximated Pareto set obtained counts 18 designs and it costs about 14,000 simulations. This procedure is similar to a manual optimization, which however in most of the cases would have produced only a reduced number of pseudo-optimal designs. The idea is to perform a first exploration in order to extract useful information about the problem. The second step is the analysis of the data obtained and their consequent exploitation through a second exploration phase, this time limited on a restricted and affordable search space.

The sequence exploration-exploitation-exploration is exactly the same hypothesized by the elitism operator of MFGA algorithm in Sect. 3.3. However that algorithm

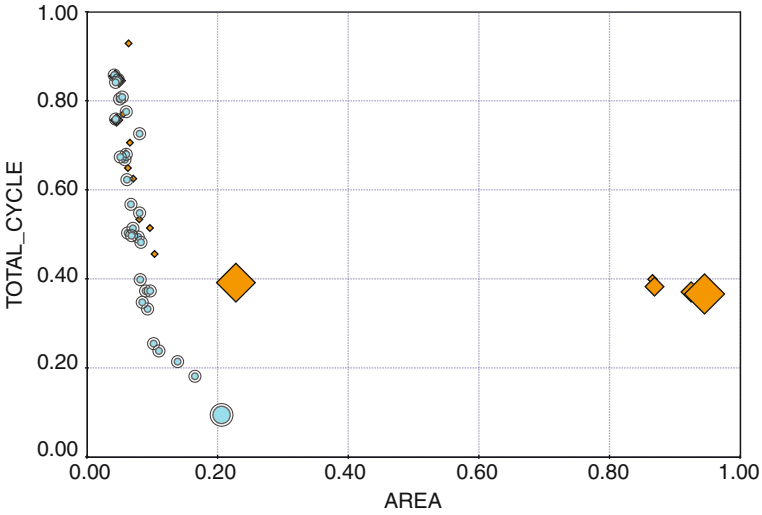

Fig. 3.7 Pareto fronts obtained with the reduced (*diamonds*) and the complete problem (*circles*). The Area and the Total Cycle objectives values are plotted on the horizontal and vertical axes, respectively. The Power Dissipation objective is represented by the size of the points: a larger point means an higher value. The highlighted points are non-dominated. The values of the metrics are normalized since the model is proprietary of STM

shows to be able of much faster convergence. An optimization run of the original problem, without the restriction introduced on the input variables, can support this observation. Indeed, MFGA after only 3,000 evaluation can produce a non-dominated set which outperforms the one obtained by the procedure just described [16]. In Fig. 3.7 the two sets are plotted on a 2D plot where the third object is represented by the size of the points.

Moreover, the analysis of the input variable values producing this new and enhanced solution set contributes to the insight in the problem structures more than the statistical calculation previously carried on. A better front is found violating the prescription proposed by this latter one.

Notwithstanding these results, the validation process cannot be considered as concluded. On the contrary, it must be a complementary tool to the research for new and better algorithms. The two processes should provide new problems each other. Once a detailed validation has stated the reliability of the optimization technique over one applicative field, new fields could open and new algorithms become necessary. Following these ideas, an industrial validation step will be described in Chap. 8.

3.5 Conclusions

The results described in the present chapter shows the expertise gained during the MULTICUBE project in handling optimization problems arising in MP-SoC architecture design. Two different but complementary meanings can be associated

to the word "handling" in this context. First, the proposed methodologies can solve the problems concretely and in a satisfactory manner: the Design Exploration tools employed within the project can support all the phases of the process, since they provide appropriate solutions to define correctly the problem, they include algorithms capable to optimize the selected metrics and they include many post-processing resources. The second meaning refers to the validation path which builds the necessary reliability to exploit the research results in an industrial context.

The definition of the problem is extremely flexible, but at the same time the fixed XML vocabulary (described in Chap. 1) is universal in the sense that all the components (tools and simulators) needed to work on the problem are able to *speak* the same language. Different simulators with different level of abstraction can be connected with the same optimization work-flow and the different optimization tools can work with all the simulators without additional modifications.

The algorithms presented are obviously the central part of the process. Although they follow different approaches, they all try to exploit the a priori knowledge of the problem structure in order to better investigate the unknown objective space shape. The presence of categorical variables is a first obstacle to overcome and indeed many of the proposed algorithms implement particular strategies for handling this kind of variables. Following this direction there is still room for improvements: is it possible, for example, to design a categorical crossover operator? It should be an operator which mixes information between the parent designs trying to maintain possible structures or good combinations among their categorical variables.

The computational cost of the simulations is another important element to analyze. The steady-state evolution implemented by some of the algorithm is a first answer. However the final number of evaluations required to achieve an accurate and uniform sample of the Pareto front is the key issue. Since all the MULTICUBE algorithms seemed to perform equivalently well, the results obtained by MFGA on the complete benchmark problem can represent a guarantee that also other algorithms can save many simulations.

Other quality of the solution set have been considered besides accuracy. Uniformity and extent are considered as complementary objectives. This opens a complete new field of research: if the result of the optimization stage is a very detailed sample of a large Pareto set, which are the points on the front that should be selected for the prototyping stage? How is it possible to help the so called Decision Maker? This stage has been considered as a separate step for long time, however recent research results in optimization tries to combine the two steps. The objective is an algorithm which returns a user-defined number of points taken from the Pareto set selecting them for their diversity.

The validation strategy proposed has a twofold merit. On one hand, simply the fact that a validation strategy has been addressed is relevant from the applicative point of view, since this is the only way of building confidence on the proposed optimization strategy. On the other hand, the validation procedure described in Sect. 3.4 contains some elements that can constitute a paradigm for evaluating optimization algorithms. A first element is to define a large set of indicators for the quality of the solution sets: a single metric can hide more than what it shows, while a deep insight in the

problem and in the algorithms can be obtained combining the results of different measurements. Another important suggestion is to check the chosen metric values at previously defined fixed numbers of evaluations. Finally, the validation stage can be considered concluded only when the new optimization algorithms have been tested against other kind of approaches (classical algorithms, manual optimization protocols, etc).

The algorithms and the procedure described in this chapter prove the overall reliability of the Design Space exploration tools, modeFRONTIER and Multicube Explorer in handling and in solving optimization problems in the field of Embedded System Design. The peculiarities of such an environment have been sufficiently recognized and exploited in order to provide solutions in affordable computational time (considering also the high consuming simulators). The study has been enough deep to open new questions for improving the capabilities of the algorithms in this field as well as for opening new research directions.

References

1. Aittokoski, T., Miettinen, K.: Efficient evolutionary method to approximate the pareto optimal set in multiobjective optimization. In: Proc. International Conference on Engineering Optimization (EngOpt) (2008)
2. Baumgartner, U., Magele, C., Renhart, W.: Pareto optimality and particle swarm optimization. IEEE Transactions on Magnetics **40(2)**, 1172–1175 (2004)
3. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga–ii. IEEE Transactions on Evolutionary Computation **6**(2), 181–197 (2002)
4. Deb, K., Goel, T.: Controlled elitist non–dominated sorting genetic algorithm for better convergence (2001). KanGal Report 200004
5. Erbas, C., Cerav–Erbas, S., Pimentel, A.: Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. IEEE Transactions on Evolutionary Computation **10**(3), 358–374 (2006)
6. Gelatt Jr., C.D., Vecchi, M., Kirkpatrik, S.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
7. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, pp. 1942–1948 (1995)
8. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: Proceedings of the Conference on Systems, Man and Cybernetics, pp. 4104–4109 (1997)
9. Metropolis, N., Rosenbluth, A., Teller, A., Teller, E.: Equation of state calculation by fast computing machines. J. Chem. Phys. **21**(1953), 1087–1092 (1953)
10. Miettinen, K.: Nonlinear Multiobjective Optimization. Kluwer Academic Publisher (1999)
11. Poloni, C., Pedirola, V.: Ga coupled with computationally expensive simulations: Tools to improve efficiency. In: Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, chap. 13. John Wiley & Sons (1998)
12. Pugh, J., Martinoli, A.: Discrete multi-valued particle swarm optimization. In: Proceedings of IEEE Swarm Intelligence Symposium, pp. 103–110 (2006)
13. Reyes-Sierra, M., Coello, C.A.: Multiple-objective particle swarm optimizers: A survey of the state of the art. http://www.lania.mx/∼ccoello/EMOO/reyes06.pdf.gz (2006)
14. Schwefel, H.: Evolution and Optimum Seeking. Wiley & Sons (1995)

15. Silvano, C., Zaccaria, V., Palermo, G.: ReSPIR: A response surface-based pareto iterative refinement for application-specific design space exploration. IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems **28**(12) (2009)
16. Turco, A., Kavka, C., Bocchio, S.: Optimization of an embedded parallel system-on-chip platform using modeFRONTIER. In: Poster Session at DATE'10 Conference (2010)
17. Turco, A., Kavka, C.: MFGA: a genetic algorithm for complex real-world optimization problems. International Journal of Innovative Computing and Applications **3**(1), 31–41 (2011)
18. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997)

Chapter 4
Response Surface Modeling for Design Space Exploration of Embedded Systems

Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, Enrico Rigoni, Carlos Kavka, Alessandro Turco, and Giovanni Mariani

Abstract A typical design space exploration flow involves an event-based simulator in the loop, often leading to an actual evaluation time that can exceed practical limits for realistic applications. Chip multi-processor architectures further exacerbate this problem given that the actual simulation speed decreases by increasing the number of cores of the chip. Traditional design space exploration lacks of efficient techniques that reduce the number of architectural alternatives to be analyzed. In this chapter, we introduce a set of statistical and machine learning techniques that can be used to predict system level metrics by using closed-form analytical expressions instead of lengthy simulations; the latter are called *Response Surface Models* (RSM). The principle of RSM is to exploit a set of simulations generated by one or more Design of Experiments strategies to build a *surrogate model* to predict the system-level metrics. The response model has the same input and output features of the original simulation-based model but offers significant speed-up by leveraging analytical, closed-form functions which are tuned during *model training.* The techniques presented in this chapter can be used to improve the performance of traditional design space exploration algorithms such as those presented in Chap. 3.

4.1 Introduction

Nowadays, Multi-Processor Systems-on-Chip (MPSoCs) and Chip-Multi-Processors (CMPs) [5] represent the *de facto* standard for both embedded and general-purpose architectures. In particular, programmable MPSoCs have become the dominant computing paradigm for application-specific processors. In fact, they represent the best compromise in terms of a stable hardware platform that is software programmable, thus customizable, upgradable and extensible. In this sense, the MPSoC paradigm minimizes the risk of missing the time-to-market deadline

V. Zaccaria (✉)
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: zaccaria@elet.polimi.it

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_4, © Springer Science+Business Media, LLC 2011

while ensuring greater efficiency due to architecture customization and software compilation techniques.

Design space exploration involves an event-based simulator in the loop. Event-based simulation still represents a fundamental tool to predict performance of candidate architectural design points. If we consider cycle-accurate system-level simulation (where the *event* corresponds to the completion of a processor clock cycle), we can have several high-complexity mathematical models to be evaluated during each event (or clock cycle), leading to an actual evaluation time that can exceed practical limits for realistic applications. Chip multi-processor architectures further exacerbate this problem given that the actual simulation speed decreases by increasing the number of cores of the chip (as shown in Fig. 4.1).

While statistical sampling techniques have already been proposed for a single simulation [12], design space exploration still lacks of efficient techniques that reduce the number of architectural alternatives to be analyzed. To face this problem, we decided to adopt statistical and machine learning techniques to create a prediction of the system level metrics for the whole simulation by using closed-form analytical expressions; the latter are called *Response Surface Models* (RSM) since they represent a suitable approximation of the system response under a specific instance of the input set of parameters (e.g. the system configuration).

This chapter is organized as follows: Sect. 4.2 presents some background on response surface models while Sect. 4.3 analyzes some of the very peculiar problems that arise when modeling embedded design spaces. Section 4.4 presents a detailed

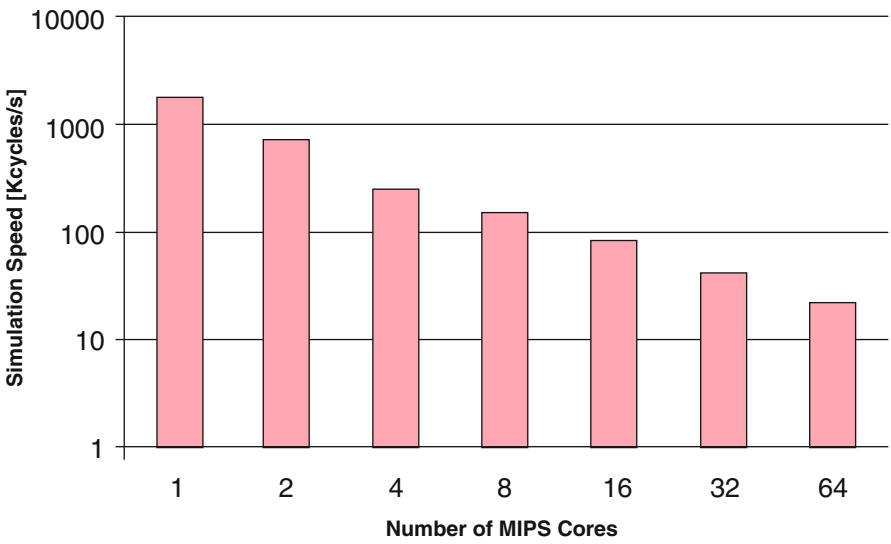

Fig. 4.1 SESC [15] simulation speed when executing the FFT kernel from [18] by varying the number of MIPS cores from 1 to 64 (Host machine: two Intel Xeons quad-core at 3 GHz)

descriptions of the RSM algorithms while Sect. 4.5 presents the general validation flow. Section 4.6 summarizes the main content of this chapter.

4.2 Background on Response Surface Models

RSM techniques are typically introduced to decrease the time due to the evaluation of a system-level metric $f(\mathbf{x})$ for each architecture configuration $\mathbf{x}$[1]. In fact, for applications of commercial interest, evaluating $f(\mathbf{x})$ can involve one or more simulations which can take several hours, depending on the platform complexity and the system resources dedicated to the simulation.

The principle of RSM is to exploit a set of simulations generated by a Design of Experiment strategy to build a *surrogate model* to predict the system-level metrics. The response model has the same input and output features of the original simulation-based model but offers dramatic speed-up in terms of evaluation since it consists of an analytical, closed-form function.

A typical RSM-based flow involves a *training phase*, in which simulation data (or *training set*) is used to tune the RSM, and a *prediction phase* in which the RSM is used to forecast unknown system response (see Fig. 4.2).

Given a system-level metric $f(\mathbf{x})$ associated with an architectural configuration $\mathbf{x}$, a response surface model $\rho(\mathbf{x})$ is defined such that:

$$f(\mathbf{x}) = \rho(\mathbf{x}) + \varepsilon \tag{4.1}$$

where ε is an ideally negligible estimation error. The prediction $\rho(\mathbf{x})$ is a function of the target architecture $\mathbf{x}$.

In simple cases, $\rho(\mathbf{x})$ may consists of a *closure*[2] of a more general function $\Phi(\mathbf{x}, \mathbf{w})$ where the vector of parameters $\mathbf{w}$ has been fixed to $\mathbf{w}_0$:

$$\rho(\mathbf{x}) = \Phi(\mathbf{x}, \mathbf{w}_0) \tag{4.2}$$

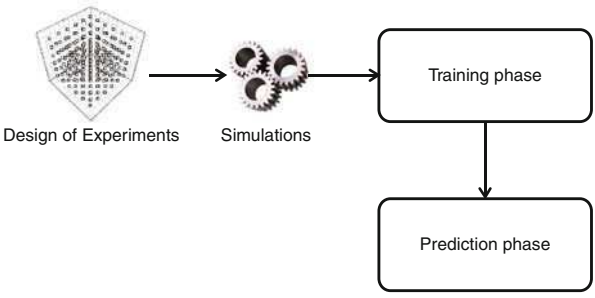

Fig. 4.2 Typical usage of a Response Surface Model

[1] Each architecture configuration is seen as a vector of configuration parameters. We use **bold** font to specify vector values.

[2] A *closure* is a first-class function with free variables that are bound in the lexical environment. Such a function is said to be "closed over" its free variables. (Source: Wikipedia).

The actual value of $\mathbf{w}_0$ is determined during the *training phase* by exploiting a set of observations $y(\mathbf{x})$ known as *training set*. The final value $\mathbf{w}_0$ is such that the estimation error:

$$\varepsilon = \Phi(\mathbf{x}, \mathbf{w}_0) - y(\mathbf{x}) \tag{4.3}$$

is minimized for both the known and future observations $y(\mathbf{x})$.

In more sophisticated cases, the structure of the function $\rho(\mathbf{x})$ is not defined a-priori but it is built by either using *neural*-like processing elements or composing elementary functions. In both cases, the training phase does not involve (only) the selection of parameters $\mathbf{w}_0$ but the navigation through a function-space to identify the optimal $\rho(\mathbf{x})$ that minimizes error ε. Of course, while prolonging the overall training process, these sophisticated model selection algorithms lend to better approximating functions $\rho(\mathbf{x})$.

4.2.1 RSM Categories

Response surface models are *surrogate models* which fit, within reasonable approximation limits, the response curve of a system with respect to the configuration $\mathbf{x}$. Being a *curve fitting* tool, RSMs can be categorized as follows:

- **Interpolation-based RSMs**. This category of curve fitting expressions is built with a constraint such that the curve $\rho(\mathbf{x})$ is equal to $f(\mathbf{x})$ for all the design points $\mathbf{x}$ belonging to the training set T:

$$\rho(\mathbf{x}) \equiv f(\mathbf{x}), \ \forall \mathbf{x} \in T \tag{4.4}$$

  while, for the remaining design points that still belong to the design space, $\mathbf{w}_0$ are calibrated such that the estimated error ε is minimal.
- **Regression-based RSMs**. This category of curve fitting expressions is such that the constraint in Eq. 4.4 does not hold; instead, the coefficients $\mathbf{w}_0$ are chosen such that a general measure of error on known training set T and the future observations is minimized.

Figure 4.3 shows a comparison between the two approaches when fitting a set of five observations of the Energy-Delay Product when varying the system cache size. As can be seen, the interpolation line (a *spline* function) passes through the observations while the regression curve (a second order polynomial) does not. Interpolation zeroes the error on the training observations but it might present an *over-fitting* effect that consists of a decreased prediction accuracy on the unknown observations. On the other hand, regression techniques, albeit with some errors on known observations, may present a greater generalization power. Nevertheless, in this book we will analyze both techniques in the domain of design space exploration.

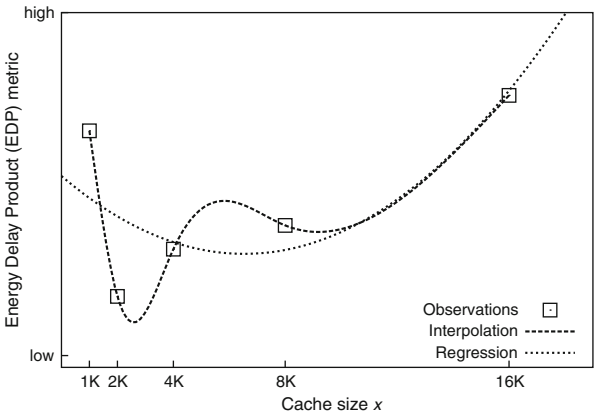

Fig. 4.3 Typical usage of a Response Surface Model

4.2.2 Design of Experiments

The training data used for identifying the parameters **w** is fundamental for creating reasonably accurate prediction models. Of course, the set should be limited given the simulation time needed to gather these data. The literature on RSM calls for a systematic Design of Experiments (DoE) [17] to identify the most suitable configurations with which the RSM can be trained. Design of Experiments is a discipline that has had a very broad application across natural and social sciences and encompassed a set of techniques whose main goal is the screening and analysis of the system behavior with a small number of simulations. Each DoE plan differs in terms of the layout of the selected design points in the design space. Several design of experiments have been proposed in the literature so far. Among the most used DoEs for training RSMs we can find:

- *Random DoE*. Design space configurations are picked up randomly by following a Probability Density Function (PDF).
- *Full Factorial DoE*. In statistics, a factorial experiment is an experiment whose design consists of two or more parameters, each with discrete possible values or "levels" and whose experimental units take on all possible combinations of these levels across all such parameters. Such an experiment allows studying the effects of each parameter on the response variable, as well as the effects of interactions between parameters on the response variable. The most important full-factorial DoE is called 2-level full factorial, where the only levels considered are the minimum and maximum for each parameter.
- *Central Composite DoE*. A Central Composite Design is an experimental design specifically targeted to the construction of response surfaces of the second order (quadratic) without requiring a three-level factorial.

It is important to note that, while factorial and central composite DoE layouts require a fixed number of points, the Random DoE can have a varying number of design points. In this book, we will leverage Random DoE for validating the proposed RSMs.

4.2.3 Over-Fitting

In statistics, over-fitting occurs when a statistical model captures systematically the random error (or *noise*) together with the underlying analytical relationship. Over-fitting generally occurs when the model complexity is excessive. This happens whenever the model has too many degrees of freedom (i.e., the size of vector **w**), in relation to the amount of data available. An over-fitting model, has poor predictive capabilities, as it can exaggerate minor sweeps in the data.

There are several methods to avoid the over-fitting risk; in the MULTICUBE project we employed techniques such as *model selection* to identify the *simplest* models (in terms of size of **w**) which can guarantee a reasonable error on the training set, and the *early stopping criterion*. The early stopping criterion consists of splitting the training data into two sets: a *training set* and a *validation set*. The samples in the training set are used to train the model, by decreasing both the error on the training data and on the validation data. The training algorithm stops as soon as the error on the validation set starts to increase.

4.3 How to Manage the Design Space of Embedded Systems

Problems emerging in the design of embedded computing systems present some characteristic features—such as the fact that all configuration parameters are discrete and possibly categorical—that deserves further discussion.

4.3.1 Discrete and Categorical Variables

SoC design problems are characterized by the fact that all configuration parameters are discrete and possibly categorical:

- Discrete variables quantify data with a finite number of values. There is a clear order relation between different values.
- Categorical (or nominal) variables classify data into categories. These are qualitative variables, in which the order of different values (categories) is totally irrelevant.

On the contrary, traditional RSM techniques usually deal with continuous design spaces. For this reasons, RSM algorithms simply ignore the discrete and/or categorical nature of variables, treating them as continuous ones.

In general this is not a pressing problem as regards discrete variables, given the order relation existing between different values. Even if the trained RSM would be able to predict the model "in the middle", this unrequested generalization will never be implemented in practice, given the discrete nature of variables in evaluation points.

Concerning categorical variables, the matter is not so simple. As a justification for pragmatically treating these variables as continuous ones, there is the fact that in many applications it is a common practice to treat categorical parameters as simple discrete ones. So if discrete variables are treated as continuous ones, the same should apply for categorical ones. But in this case there is a non-negligible difference: there is no order relation between different variables values. For this reason, caution is requested: the training database should be examined, in order to determine, from case to case, if it is possible to treat categorical parameters as continuous ones. One criterion of decision could be to consider if different subsets corresponding to different categories present analogies (correlations) in response behavior. In case of positive answer, it probably makes sense to train the RSM profitably on the full database, taking advantage of the simple reduction to a continuous domain. If this is not the case, one possible solution could be to treat these different categories as separate sub-problems: different RSMs should be trained separately for each category.

As a final remark, in the design of embedded systems, many discrete input variables are power of two (e.g., memory size): $x = 2^m$. In these cases, it is convenient to perform a variable transformation, taking the exponent m as the actual input parameter, and considering x as a dependent (auxiliary) variable. In this way, the discrete parameter presents two characteristics that are desirable from the point of view of any RSM training algorithm: its values are equispaced and it is well scaled (as regards its range of variation).

4.3.2 Optimal DoE

The following considerations are usually found when dealing with Radial Basis Functions (where they have a straightforward formulation), but they can be generalized for all RSM algorithms. The generic application refers to multivariate scattered training data in a continuous design space.

A generic set of scattered training points $\{\mathbf{x}_i\,,\ i = 1, \ldots, n\}$ is characterized by two quantities: the *fill distance* h, and the *separation distance* q. These quantities, defined in the followings, are shown in Fig. 4.4.

The fill distance h is defined as the radius of the largest inner empty disk:

$$h = \max_{\mathbf{x}\in\Omega} \min_{1\le j\le n} \|\mathbf{x} - \mathbf{x}_j\|, \tag{4.5}$$

where $\Omega \subset \mathbb{R}^d$ is the domain of definition of $f(\mathbf{x})$. In order to achieve better approximation quality of the RSM one should minimize the fill distance: min$h$.

The separation distance q is defined as the minimum distance between two training points:

$$q = \min_{i\neq j} \left\{ \|\mathbf{x}_i - \mathbf{x}_j\| \right\}. \tag{4.6}$$

In order to improve the numerical stability of the RSM training algorithm, one should maximize the separation distance: max$q$. Therefore, to improve both approximation

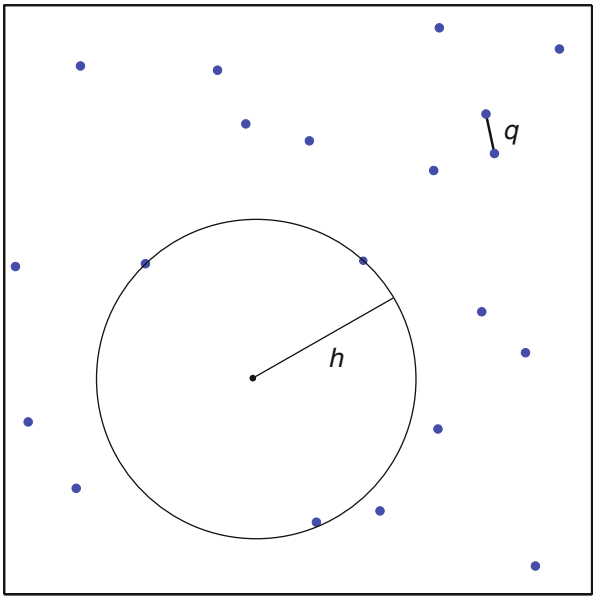

Fig. 4.4 Fill distance h and separation distance q

quality and numerical stability, one should maximize the ratio $\max(q/h)$. Clearly this objective is achieved for a well distributed, almost uniform, set of training points. But in general, for scattered data, one deals with $q \ll h$. In case of uniform distribution of data, there is no way for further improving both objectives: there is a trade-off situation between $\min h$ and $\max q$. This fact explains the choice of Sect. 4.2.2: *Random DoE* (i.e., a uniform Monte Carlo) is a good algorithm for generating points to be used as training database by RSM, since it maximizes the ratio $\max(q/h)$. In general any space filler DoE can be used.

Some enhanced techniques could be implemented in case a even more accurate uniformity is necessary. E.g., *Incremental Space Filler*, for augmenting an existing database in order to fill the space in a uniform way (filling the gaps), or *Uniform Latin Hypercube* that uses Latin Hypercube - an advanced Monte Carlo which maps better the marginal probability distribution of each single variable—for generating random numbers conforming to a uniform distribution.

In case of discrete variables, the separation distance q has a lower bound, strictly related to the variables resolution (number of levels). In general numerical stability is not a pressing problem. On the contrary, in order to achieve approximation quality, it is important to fill as uniformly as possible the gaps in the regular grid formed by the set of all the combinations of admissible discrete variables values. Usually space filling techniques, even though conceived for continuous design space, are able to manage correctly also discrete variables. So in general neither approximation quality is an insurmountable problem, given that the seek for uniformity in a discrete design space is properly considered.

4.3.3 Pre-Processing and Scaling Data

Data used to derive analytical models, also if originated from the same source/modeled architecture, due to different positions in the design space can have different values distribution and orders of magnitudes. Especially when the case is the latter, to create better prediction it is better to *pre-process* and/or *scale* data.

Data transformation is very important because in most of the cases the analytical models used to predict the data work better when data distribution follows some rules. As an example, if we consider an analytical model that uses the standard deviation of the training data to predict unknown data, this standard deviation values can be very high if the data distribution is skewed. In this case, it is highly recommended to first transform the data to approach a better symmetry and then to perform the model training and related prediction.

Box-Cox power transformation. A powerful transformation adopted in the above-mentioned cases is called Box-Cox power transformation [4]. The Box-Cox power transformation is a useful data pre-processing technique used to reduce data variation, make the data more normal distribution-like and improve the correlation between variables. The power transformation is defined as a continuously varying function, with respect to the power parameter λ:

$$y_k^{(\lambda)} = \begin{cases} (y_k^{\lambda} - 1)/\lambda, & \text{if } \lambda \neq 0 \\ \log y_k, & \text{if } \lambda = 0 \end{cases} \tag{4.7}$$

In the validation results of the models that we adopted in this book, we considered a family of transformations as potential candidates $\lambda\{1, 0.5, 0, -1\}$. All the Box-Cox power transformations are only defined with positive values. In case of negative values, a constant value has to be added in order to make them positive. To keep the prediction consistent with the actual objective functions of the target problem, an inverse Box-Cox transformation has been applied on the predicted data.

Some care has to be taken when performing the inverse Box-Cox transformation, since the condition $\lambda\, y_k^{(\lambda)} + 1 > 0$ (when $\lambda \neq 0$) has to be satisfied. Therefore possible unfeasible outcomes has to be taken into account.

Centering and scaling data. Another pre-processing step that is usually applied to the data after the data transformation is the centering and scaling step. The goal of this step is to remove the bias from the input data (mean equal to zero) and to standardize the variance (standard deviation equal to 1). This transformation is also called "Autoscaling". When the *autoscaling* transformation is applied to a set of data, from each value the mean value is removed and it is scaled by the standard deviation: $y_{\text{autoscaled}} = (y_{\text{original}} - \mu_y)/\sigma_y$. Another common alternative step is to normalize data in the unitary interval $[0, 1]$: $y_{\text{normalized}} = (y_{\text{original}} - \min y)/(\max y - \min y)$. Usually this normalization step is performed on input variables too: in this way data result to be well scaled, being perfectly comparable as regards range extensions. This solution prevent numeric issues that usually arise during RSM training in presence of different scaled variables.

4.4 Algorithms Description

This section describes the fundamental concepts of the RSMs used in the Multicube design flow. All the presented RSMs have been integrated either in the Multicube Explorer open source tool and/or in the modeFRONTIER design tool. The set of RSMs consists of the following models:

- Linear regression (Regression-based).
- Splines (Regression-based).
- RBF (Interpolation-based).
- Neural Networks (Regression-based).
- Kriging (Interpolation/Regression-based).
- Evolutionary Design (Regression-based).

In the following paragraphs we will describe each model in detail, while some results of the application of RSM techniques to some industrial case studies will be reported in Chap. 8.

4.4.1 Linear Regression

Linear regression is a technique for building and tuning an analytic model $\rho(\mathbf{x})$ as a linear combination of $\mathbf{x}$'s parameters in order to minimize the prediction residual ε.

We apply regression by taking into account also the interaction between the parameters and the quadratic behavior with respect to a single parameter. We thus consider the following general model:

$$\Phi(\mathbf{x}, \mathbf{w} = [\mathbf{a}, \mathbf{b}, \mathbf{c}]) = a_0 + \sum_{j=1}^{n} a_j x_j^2 + \sum_{l=1}^{n} \sum_{j=1, j \neq l}^{n} b_{j,k} x_l x_j + \sum_{j=1}^{n} c_j x_j \qquad (4.8)$$

where x_j is the *level* (numerical representation) associated with the j-th parameter of the system configuration, while $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are a decomposition of the RSM family parameters $\mathbf{w}$ and n is the number of parameters of the design space.

Least squares analysis can be used to determine a suitable approximation of $\mathbf{w}$. The least squares technique determines the values of unknown quantities in a statistical model by minimizing the sum of the squared residuals (the difference between the approximated and observed values).

A measure of the *quality of fit* associated with the resulting model is called *coefficient of determination* and defined as follows:

$$R^2 = \frac{\sum_k (y_k - \bar{y})^2}{\sum_k (\rho_k - \bar{y})^2}. \qquad (4.9)$$

where y_k is the k-th observation, $\bar{y}$ is the average of the observations, and ρ_k is the prediction for the y_k observation. R^2 corresponds to the ratio of variability in a data

set that is accounted for by the statistical model. Usually, the higher R^2 the better is the quality of fit (with $0 \leq R^2 \leq 1$).

Although adding parameters to the model can improve the R^2, there is a risk of exceeding the actual information content of the data, leading to arbitrariness in the final model parameters (also called *over-fitting*). This reduces the capability of the model to generalize beyond the fitting data, while giving very good results on training-data. In particular, this phenomenon occurs when the model is excessively complex in relation to the amount of data available. A model which has been over-fit, generally has poor predictive performance, as it can exaggerate minor fluctuations in the training data.

To this purpose, we introduce an 'adjusted' definition of the R^2. This terms adjusts for the number of explanatory terms in a model; it increases only if the terms of the model improve it more than expected by chance and will always be less than or equal to R^2; it is defined as:

$$1 - (1 - R^2)\frac{N - 1}{N - p} \tag{4.10}$$

where p is the total number of terms in the linear model (i.e., the set of coefficients a, b, c), while N is sample set size. Adjusted R^2 is particularly useful in the *model selection* stage of model building.

Linear regression model selection. In order to understand the optimal number of terms of the linear model and the corresponding model order, we analyzed the behavior of the RSM cross-validation error and adjusted R^2 by varying the number of random training samples (derived from the simulations of the target applications). Equation 4.10 represents an improved measure of the overall *quality of fit* of the linear regression: it is inversely proportional to the model's *degrees of freedom* (i.e., $N - p$) which, in turn, depend on the order of the chosen polynomial $\rho(\mathbf{x})$. As a matter of fact, higher degrees of freedom increase the chance of reduced variance of the model coefficients thus improving model stability while avoiding over-fitting [9, 10]. In this book, our heuristic model selection tries to maximize the number of degrees of freedom and, at the same time, to minimize (in the order of 200) the number of simulations needed to build the model. Thus, as a "rule of thumb", we set a maximum number of terms (to increase the chance of good quality of fit) and eventually, we limit the set of considered models to the following configurations:

1. First order model, without any interaction between parameters.
2. First order model, with interaction between parameters.
3. Second order model, without any interaction between parameters.

4.4.2 Radial Basis Functions

Radial basis functions (RBF) represent a widely used interpolation/approximation model [13]. The interpolation function is built on a set of training configurations $\mathbf{x}_k$

as follows:

$$\Phi(\mathbf{x}, \mathbf{w}) = \sum_{k=1}^{N} w_k \, \gamma(\|\mathbf{x} - \mathbf{x}_k\|) \tag{4.11}$$

where γ is a scalar *distance* function, w_k are the weights of the RBF and N is the number of samples in the training set. In this paper, we consider the following definitions for γ

$$\gamma(z) = \begin{cases} z & \text{linear} \\ z^2 \log z & \text{thin plate spline} \\ (1+z^2)^{1/2} & \text{multiquadric} \\ (1+z^2)^{-1/2} & \text{inverse multiquadric} \\ e^{-z^2} & \text{gaussian} \end{cases} \tag{4.12}$$

The weights w_k are the solution of a matrix equation which is determined by the training set of configurations $\mathbf{x}_k$ and the associated observations y_k:

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ A_{21} & A_{22} & \dots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \dots & A_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{4.13}$$

where:

$$A_{jk} = \gamma(\|\mathbf{x}_j - \mathbf{x}_k\|), \qquad j, k = 1, 2, \dots, N, \tag{4.14}$$

4.4.3 Splines

Spline-based regression has been recently proposed by Lee and Brooks [7] as a powerful method for the prediction of power consumption and performance metrics. A spline RSM is composed of a number of piecewise polynomials σ_L associated with each of the parameters x_j of the architecture:

$$\Phi(\mathbf{x}, \mathbf{w} = [\mathbf{a}, \mathbf{b}, \mathbf{k}]) = a_0 + \sum_{j=1}^{N} a_j \sigma_L(x_j, \mathbf{b}, \mathbf{k}) \tag{4.15}$$

The piece-wise polynomial σ_L is divided into L intervals defining multiple different continuous polynomials with endpoints called *knots*. The number of knots L can vary depending on the amount of available data for fitting the function and the number of levels associated with parameter x_j, but more knots generally lead to better fits. In fact, relatively simple linear splines may be inadequate for complex, highly

non-linear relationships. In the context of the project, we exploited *restricted* cubic splines (RCS) where the expressions associated with σ are third order polynomials. Restricted cubic splines are such that the first and second order derivative at the knots is the same for *adjacent* polynomials while they present a linear behavior on the *tails*.

As an example, a closed form expression of a RCS with three knots ($L = 3$) is the following:

$$\sigma_3(x_j, \mathbf{b}, \mathbf{k}) = b_{0,j} + b_{1,j}x_j + b_{2,j}x_j^2 + b_{3,j}x_j^3 + b_{4,j}(x_j - k_1)^3 + b_{5,j}(x_j - k_2)^3 + b_{6,j}(x_j - k_3)^3$$

where k_j are the knot points in the function domain.

To determine the number of knots, we started by considering that five knots or fewer are generally sufficient for restricted cubic splines. While fewer knots may be required for small data sets, with a large number of knots increases the risk of over-fitting the data. In particular, we adopted the following policies for the selection of the number of knots depending on the number of levels of the j-th parameter x_j:

- If the number of levels is greater than 5, $L = 5$.
- If the number of levels is smaller than 3, $L = 0$ (the spline is a linear function of the parameter).
- Otherwise, the number of knots is equal to the number of levels.

4.4.4 Neural Networks

For function approximation purposes, Feed-forward Neural Networks (also known as Multilayer Perceptrons) are a very efficient and powerful tool. Feed-forward networks are organized in successive layers of neurons. The data flow is unidirectional: the data pass from the first *input* layer to the last *output* layer, and are elaborated incrementally by the intermediate *hidden* layers.

The model of each single neuron is straightforward:

$$u = \sum_{i=1}^{n} w_i\, x_i + b \qquad y = f(u) = f\left(\sum_{i=1}^{n} w_i\, x_i + b\right) \tag{4.16}$$

The *net input* u is a linear combination of the input values x_i: the *weights* w_i and the *bias* b (or *firing threshold*) represent the free parameters of the model. The net input is transformed by means of a *transfer function* f (or *activation function*)—that in general is non linear—giving the neuron's output y. The behavior and the complexity of the network are defined by the way its neurons are connected: so in general the model $\Phi(\mathbf{x})$ is a complex recursive function that is usually regarded as a black-box, with no given explicit analytical expression.

It has been shown in [3] that Neural Networks (NN) with one single non-linear hidden layer and a linear output layer are sufficient for representing any arbitrary (but

sufficiently regular) function. The necessary condition is a sufficiently high number of neurons in the hidden layer.

NNs learn by example: given a training data set of inputs and a set of targets as outputs, the network's weights and biases are adjusted in order to minimize errors in its predictions on the training data. The best known training algorithm is back-propagation: a very effective approach consists of the Levenberg–Marquardt algorithm, as outlined in [2].

The initialization of the weights of NN for training is usually set at random small values. However the Nguyen-Widrow initialization technique greatly reduces the training time (see [11]).

In this book, we use classical Feed-forward NN with one hidden layer: this layer has a sigmoid transfer function, while the output layer has a linear transfer function. Back-propagation training algorithm is used, in particular the Levenberg–Marquardt algorithm is implemented, with a fixed number of iterations. The Nguyen-Widrow initialization technique is also implemented. The training data are internally normalized, both in input and in output, in order to exploit the natural range of definition of transfer functions.

In this context, there is only one free parameter to be set: the number of neurons in the hidden layer. Automatic network sizing has been implemented, using the value proposed in [16].

4.4.5 Kriging

Kriging is a very popular regression methodology based on Gaussian Processes [14]. This RSM algorithm can be interpolating or approximating, depending if a noise parameter is set to zero or to nonzero values.

Kriging is a Bayesian methodology (named after professor Daniel Krige), used as a main tool for making previsions employed in geostatistics, e.g., for soil permeability, oil and other minerals extraction, etc. (originally it was developed for predicting gold concentration at extraction sites). The formalization and dissemination of this methodology is due to Professor Georges Matheron [8], who indicated the Krige's regression technique as *krigeage*.

The Kriging estimator is a linear estimator, i.e., the estimated value is expressed as a linear combination of the training values, in other words:

$$\Phi(\mathbf{x}) = \sum_{k=1}^{N} \lambda_k(\mathbf{x})\, y_k \tag{4.17}$$

where the weights $\lambda_1, \ldots, \lambda_N$, are obviously point-dependent.

Kriging can also produce an estimate of the error, i.e., a prediction value and an expected deviation from the prediction.

The Kriging behavior (smoothness of the model) is controlled by a covariance function, called the variogram function, which rules how varies the correlation between the response values in function of the distance between different points. A function can be rougher or smoother, can exhibit large or small ranges of variation, can be affected by a certain amount of noise, and all these features can be resumed in a variogram model.

More precisely, the covariance function $\mathrm{Cov}(\mathbf{x}_1, \mathbf{x}_2)$ only depends on the distance between two points:

$$\mathrm{Cov}(\mathbf{x}_1, \mathbf{x}_2) = \sigma - \gamma(\|\mathbf{x}_1 - \mathbf{x}_2\|) \tag{4.18}$$

where $\gamma(h)$ is the variogram function, and σ is the sill, i.e., the asymptotic value of γ.

There are several variogram types that can be employed: Gaussian, Exponential, Matèrn, Rational Quadratic. Usually Gaussian is the first (default) choice: the generated metamodel is infinitely differentiable.

Each variogram function is characterized by three different parameters: range, sill, and noise.

The variogram range of the covariance function corresponds to a characteristic scale of the problem. If the distance between two points is larger than the range the corresponding outcomes should not influence each other (completely uncorrelated). Range is inversely related to the number of oscillations of the function. Small ranges mean sudden variations, while large ranges mean very regular trends, with very few oscillations.

The variogram sill corresponds to the overall variability of the function. The gap between the values of very distant points should be of the same scale of magnitude of the sill.

Variogram noise can also be tuned to fit the expected standard error in the observations. Larger amount of noise will result in smoother responses, while zero noise means exact interpolation.

Parameters determination can be based on previous knowledge on similar problems, or may be guessed by following two automatic fitting strategies: maximizing the Likelihood of the model given the training dataset or maximizing the Leave-One-Out (LOO) Predictive Probability. The Likelihood of the variogram is the probability that a statistical distribution associated to the variogram parameters could generate the given dataset. Likelihood is larger for good fitting models, but penalizes unnecessarily complex models. Maximum likelihood models are the smoothest models with best agreement to the dataset. The Leave-One-Out Predictive Probability gives a measure of the goodness of the model also removing one point at a time in the dataset and estimating the value at the removed site on the basis of the remaining designs. Models predicting the smallest errors at "difficult" points are rewarded with high LOO Predictive Probability. Maximum LOO Predictive Probability privileges good fitting models which do not loose prediction performance by removing some design. However, computation of the LOO Predictive Probability can be quite intensive, more than the computation of the Likelihood.

4.4.6 Evolutionary Design

Evolutionary Design (ED) [1] is an effective implementation of Genetic Programming (GP) methodology [6] for symbolic regression.

In general the goal of the regression task is to discover the relationship between a set of inputs, or independent variables $\mathbf{x}$ given an observable output, or dependent variable y. In standard regression techniques the model functional form $\Phi(\mathbf{x}, \mathbf{w})$ is known beforehand. The only unknown values are some coefficients $\mathbf{w}$, i.e., the free parameters of the model.

ED uses low-level primitive functions. These functions can be combined to specify the full function. Given a set of functions, the overall functional form induced by genetic programming can take a variety of forms. The primitive functions are usually standard arithmetical functions such as addition, subtraction, multiplication and division but could also include trigonometric and transcendental functions. Any legal combination of functions and variables can be obtained.

Each individual in GP corresponds to a given mathematical expression, and it is represented by means of a parse tree. For example, the expression

$$f(x, y) = 2xy - (x + 1)^2 \tag{4.19}$$

is represented by the parse tree depicted in Fig. 4.5.

Symbolic regression is then a composition of input variables, coefficients and primitive functions such that the error of the function with respect to the expected output is minimized. Shape and size of the solution is not specified before the regression. Number of coefficients and their values are issues that are determined in the search process itself. By the use of such primitive functions, genetic programming is in principle capable of expressing any functional form that use the primitive functions provided by the user. Unlike the traditional methods, the Evolutionary Design process automatically optimizes both the functional form and the coefficient values.

ED is capable of providing answers in the symbolic language of mathematics, while others methods only provide answers in the form of sets of numbers, weights, valid in the context of a model defined beforehand. So, after the training, the explicit formula of the regression model is promptly available.

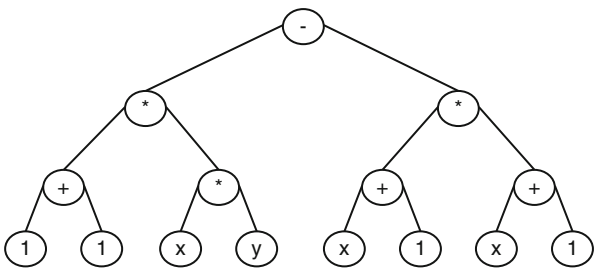

Fig. 4.5 Parse tree representing the mathematical expression $f(x, y) = 2xy - (x + 1)^2$

4.5 General Validation Flow of RSMs

To verify the quality of the predictions generated by a RSM we will compare the predicted metric values with the actually observed ones to obtain then a quality index for the model. To do so, we introduce the *average normalized error*:

$$\eta = \frac{\sum_{i \in \Theta} \left(\frac{\sum_{\mathbf{x} \in \Delta} \left| \frac{\rho(\mathbf{x})_i - y(\mathbf{x})_i}{y(\mathbf{x})_i} \right|}{|\Delta|} \right)}{|\Theta|} \tag{4.20}$$

where:

- Θ is the set of system metrics,
- Δ is the set containing all the design space points,
- $y(\mathbf{x})_i$ is the actual value of the metric $i \in \Theta$ for the design space point $\mathbf{x} \in \Delta$, and
- $\rho(\mathbf{x})_i$ is the estimated value of metric $i \in \Theta$ for the design space point $\mathbf{x} \in \Delta$.

An appropriate RSM should present a behavior where the greater is the training set size, the better is the average normalized error. As a matter of fact, we will verify that the error decreases growing the training set size, and this is done re-running the model construction with a greater training sets and calculating the average normalized error.

If random processes are involved during the selection of the design space points used as training set, the prediction results can present variability (especially for training sets with small size with respect to the design space). To characterize this effect, the validation methodology will be repeated to identify a set of stable statistical properties.

4.6 Conclusions

In this chapter, we have introduced a set of statistical and machine learning techniques that can be used to improve the performance and/or accuracy of design space exploration techniques by predicting system level metrics without resorting to long simulations. The presented techniques (called Response Surface Models) leverage a set of closed-form analytical expressions to infer an approximation of the actual system response by either exploiting regression or interpolation modeling. Results of the validation of RSM algorithms will be described in Chap. 8.

References

1. Fillon, C.: New strategies for efficient and practical genetic programming. Ph.D. thesis, Università degli Studi di Trieste (2008)

2. Hagan, M.T., Menhaj, M.B.: Training feedforward networks with the marquardt algorithm. IEEE Trans. on Neural Networks **5**(6) (1994)
3. Irie, B., Miyake, S.: Capabilities of three-layered perceptrons. In: Proceedings of the IEEE International Conference on Neural Networks, pp. I–641 (1998)
4. Joseph, P., Vaswani, K., Thazhuthaveetil, M.: Construction and use of linear regression models for processor performance analysis. High-Performance Computer Architecture, 2006. The Twelfth International Symposium on pp. 99– 108 (2006)
5. Keutzer, K., Malik, S., Newton, A.R., Rabaey, J., Sangiovanni-Vincentelli, A.: System level design: Orthogonolization of concerns and platform-based design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **19**(12), 1523–1543 (2000)
6. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
7. Lee, B.C., Brooks, D.M.: Accurate and efficient regression modeling for microarchitectural performance and power prediction. Proceedings of the 12th international conference on Architectural support for programming languages and operating systems **40**(5), 185–194 (2006). DOI http://doi.acm.org/10.1145/1168917.1168881
8. Matheron, G.: Les variables régionalisées et leur estimation: une application de la théorie des fonctions aléatoires aux sciences de la nature. Masson, Paris (1965)
9. Montgomery, D.C., Runger, G.C.: Applied Statistics and Probability for Engineers. Wiley (2006)
10. Montgomery, D.C.: Design and Analysis of Experiments. John Wiley and Sons (2005)
11. Nguyen, D., Widrow, B.: Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: Proceedings of IJCNN, vol. 3, pp. 21–96 (1990)
12. Perelman, E., Hamerly, G., Biesbrouck, M.V., Sherwood, T., Calder, B.: Using simpoint for accurate and efficient simulation. In: ACM SIGMETRICS Performance Evaluation Review, pp. 318–319 (2003)
13. Powell, M.J.D.: The theory of radial basis functions. In: Advances in Numerical Analysis II: Wavelets, Subdivision, and Radial Basis Functions, W. Light (ed, pp. 105–210. University Press (1992)
14. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
15. Renau, J., Fraguela, B., Tuck, J., Liu, W., Prvulovic, M., Ceze, L., Sarangi, S., Sack, P., Strauss, K., Montesinos, P.: SESC simulator (2005). Http://sesc.sourceforge.net
16. Rigoni, E., Lovison, A.: Automatic sizing of neural networks for function approximation. In: SMC2007, pp. 2005–2010 (2007)
17. Santner, T.J., B., W., W., N.: The Design and Analysis of Computer Experiments. Springer-Verlag (2003)
18. Woo, S., Ohara, M., Torrie, E., Singh, J., Gupta, A.: Splash-2 programs: characterization and methodological considerations. Proceedings of the 22th International Symposium on Computer Architecture p. 2436 (1995)

Chapter 5
Design Space Exploration Supporting Run-Time Resource Management

Prabhat Avasare, Chantal Ykman-Couvreur, Geert Vanmeerbeeck, Giovanni Mariani, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria

Abstract Running multiple applications optimally in terms of Quality of Service (e.g. performance and power consumption) on embedded multi-core platforms is a huge challenge. Moreover, current applications exhibit unpredictable changes of the environment and workload conditions which makes the task of running them optimally even more difficult. This dynamic trend in application runs will grow even more in future applications.

This Chapter presents an automated tool flow which tackles this challenge by a two-step approach: first at design-time, a *Design Space Exploration* (DSE) tool is coupled with a platform simulator(s) to get optimum operating points for the set of target applications. Secondly, at run-time, a lightweight *Run-time Resource Manager* (RRM) leverages the design-time DSE results for deciding an operating configuration to be loaded at run-time for each application. This decision is performed dynamically, by taking into consideration available platform resources and the QoS requirements of the specific use-case. To keep RRM execution and resource overhead at minimum, a very fast optimisation heuristic is integrated.

Application of this tool-flow on a real-life multimedia use case (described in Chap. 9) will demonstrate a significant speedup in optimisation process while maintaining desired Quality of Service.

5.1 Introduction

The future of embedded computing is shifting to multi-core designs due to handling of simultaneous multiple applications and at the same time to boost performance given the unacceptable power consumption and operating temperature increase of fast single-core processors. This introduces new big challenges for application and platform designers. The first challenge is the support for a variety of applications:

P. Avasare (✉)
IMEC, Leuven, Belgium
e-mail: avasare@imec.be

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_5, © Springer Science+Business Media, LLC 2011

mobile communications, networking, automotive and avionic applications, multimedia. These applications may run concurrently, start, and stop at any time. There is lot of dynamism in these applications which makes it challenging to run them optimally. Each application may have multiple configurations, with different constraints imposed by the external world or the user (deadlines and quality requirements, such as audio and video quality, output accuracy), different usages of various types of platform resources (processing elements, memories and communication), and different costs (performance, power consumption, bandwidth). The second challenge is the platform heterogeneity, happening between platforms and within a platform. Even for similar platforms, process variation endows them with different performance characteristics. Furthermore the resource requirement for each application may vary over time (by the time applications are put in the market for sell) due to hardware adaptation to physical constraints (power, temperature, battery life, and aging). Hence, it is untenable to ask software vendors to adapt or optimize their applications for each platform. The third challenge is time to market, which makes software development productivity of paramount importance. Designers are getting much less time to design their application and platforms whereas the complexity of the design and mapping is increasing. This can make it hard to perform exhaustive design space explorations. At the same time, due to the increasing complexity of the architecture, it is almost impossible for traditional design techniques to identify an optimal static configuration matching the available architecture resources with the dynamic requirements (i.e., performance and power consumption).

Given such tough challenges, one can design an embedded system by making conservative worst-case assumptions, but that will not lead to an efficient design. Especially in embedded system design, where there are very tight constraints (e.g. cost of a device), it is not acceptable to have a device with large inefficiency. To overcome these challenges, we have developed a methodology to enable efficient running of multiple applications on multi-core platforms. Our approach uses a run-time manager to control different possible application configurations [39]. Thus, concerning all the parameters that can be changed at run-time (e.g. the number of cores used to run an application and their operating frequencies), a run-time manager layer is built to achieve desirable *Quality of Service* (QoS) for the applications.

Our run-time manager layer consists of a Run-time Resource Manager (RRM) with following features:

- It supports a holistic view of the resources. This is needed for global resource allocation decisions, arbitrating between all applications, and minimizing the total costs.
- It transparently optimizes the resource usage and the application mapping on the platform. This is needed to facilitate the application development and mapping from diverse application domains.
- It dynamically adapts to changing environment. This is needed to enable the best usage of resources and to achieve a high efficiency under changing environment and requirements. To that end, dynamic resource allocation and dynamic reconfiguration of applications must be supported. Also, quality requirements

and resources must be adapted dynamically (e.g. by adjusting the processor clock frequency, or by switching off some processors) in order to control platform performance (i.e., the power consumption and the heat dissipation of the platform).
- Such a resource management problem is a Multi-Objective Optimization (MOO) problem (known as Multi-dimension Multiple-choice Knapsack Problem or MMKP) falling into a complexity category of NP-Hard problems [11]. Since our RRM is intended for embedded platforms, it is implemented as a lightweight heuristic implementation which consumes during its execution as little platform resources as possible and does not impact heavily execution time of the running application.

With respect to discussions in this Chapter, run-time manager consists of only RRM. Hence we have used term run-time manager and RRM interchangeably.

To alleviate run-time decision making (i.e., to reduce computational complexity at run-time) and to avoid conservative worst-case assumptions, our run-time management methodology consists of two phases:

- First, a design-time Design Space Exploration (DSE) per application derives a multi-dimensional Pareto set of optimal configurations on a given multi-core platform. During this phase, optimization and modeling techniques presented in Chaps. 3 and 4 are adopted.
- Second, a low-complexity Run-time Resource Manager (RRM) is incorporated on top of the basic services of the platform Operating System (OS) and is acting as an exception handler at run-time to optimize the usage of resources.

In the first phase, at design time, each configuration is characterized by a code version together with an optimal combination of constraints, used resources, and costs. The different code versions refer to different parallelizations of the application into parallel tasks and data transfers to shared and local memories. To enable the design-time DSE phase, the methodology presented in this Chapter considers only applications within a set defined at design-time. System-wide approaches, for resource management at coarse grain, considering software applications for which the platform was not directly designed (as the approach proposed in [4]), are discussed in Chap. 6.

In the second phase of run-time management presented in this Chapter, whenever the environment is changing dynamically (e.g. when a new application or use case starts, or when the user requirements change), RRM reacts as follows:

- It selects a configuration from the Pareto set of each active application, according to the available resources, in order to minimize the costs, while satisfying the constraints.
- Second, it reconfigures and maps the application on the platform i.e., it assigns the platform resources, it adapts the platform parameters, it loads the application tasks, and it issues the application executions according to the newly selected configurations.

The remainder of this Chapter is organized as follows. Section 5.2 overviews the state-of-the-art on RRM for embedded multi-core platforms and details RRM problem tackled in this Chapter. Section 5.3 formulates the RRM problem. Section 5.4 presents the exploration tool flow to solve this RRM problem. Conclusions are drawn in Sect. 5.5.

5.2 Run-Time Resource Management in Embedded Systems

This section introduces state-of-the-art techniques on RRM and places our methodology in comparison to previous literature.

In the context of Run-time Resource Management (RRM), traditional approaches can be roughly classified into either pure design-time approaches or pure run-time approaches. Nevertheless, they suffer from the following drawbacks:

- First, some of them are applicable only for single-core platforms [32], or for homogeneous multi-core platforms [42], but not for heterogeneous multi-core platforms.
- Second, none of the existing approaches proposes a complete framework. They are based only on task scheduling, i.e., on task ordering and assignment. A good overview of available design-time algorithms can be found in [28]. Some others are based only on slowing or shutting down the platform resources [5] and on Dynamic Voltage and Frequency Scaling (DVFS) [9, 14, 19, 27].
- Third, the objective of the majority of these approaches is performance optimization [3, 6, 8, 15, 18], and not together with power consumption optimization.
- Finally, design-time approaches involve slow heuristics [27, 29, 30] using Integer Linear Programming (ILP) and cannot be used at run time. On the other hand, to reach a lightweight implementation, run-time approaches hide the specification of the internal application tasks, and they do not fully exploit the task mapping choices of the target platform. Hence these approaches are sub-optimal.

Hence neither the existing pure design-time approaches nor the existing pure run-time approaches are efficient to solve this complex RRM problem. To alleviate the run-time decision making and to avoid worst-case assumptions, new research directions are ongoing and propose a mixed design-time and run-time approach:

- The Task Concurrency Management (TCM) methodology [34–36], explores the energy-performance trade-offs at the system level. To reach an efficient usage of the platform resources, this methodology models the application at a finer granularity than traditional task graphs. It identifies the sub-tasks of the application that can run in parallel on a heterogeneous multi-processor platform. It also includes data access and memory management at the task level [20, 39].
- Scenario-based approaches [12, 23], which are based on the concept of application scenarios identified at design time, operate as follows. First, a profiling-based analysis of various run-time situations of the application is performed. Then,

these run-time situations are clustered into a few dominant application scenarios and a backup scenario. At run time, the actual scenario is detected with a simple detector, and the application is executed with the configuration decided for that scenario at design time.

- The task scheduling techniques [34, 35] schedule one task at a time, making use of the entire platform for each task. This leads to inefficient usage of the platform. The techniques in [31, 37] allow parallel execution of the tasks, while sharing the platform resources. But they assume that all tasks start at the same time. This assumption can lead to idle platform processors until the next RRM call. These techniques are extended in [24], which allows overlapped sharing of the platform resources. The latter technique considers the start time and the periodic information present in applications such as frame processing in video decoding, or packet processing in wireless applications.
- RRM for multi-core embedded platforms [16, 17] also exploits the number of available cores, in addition to voltage and frequency. Different parallelized versions of a single application are used to trade-off the available platform resources with the performance and the power consumption.
- The addition of the parallelism to the set of platform parameters significantly increases the design space of operating modes. Innovative and efficient techniques for RRM are needed to extend the traditional approaches for power consumption optimization. Recent studies in this field address the problem by modeling it as a Multi-dimension Multiple-choice Knapsack Problem (MMKP) and solving it through dedicated heuristics [31, 37]. Another approach [22] proposes a run-time management technique for task-level parallelism in order to optimize the performance under a power consumption budget.
- Advanced technologies such as sub-45 nm CMOS and 3D integration are known for the increased number of reliability failure mechanisms. Nevertheless, classical reliability-aware approaches are no longer viable, since they propose ad-hoc failure or worst-case solutions, which incur a significant cost penalty. In [26], the state-of-the-art in reliability management techniques is summarized, and a new proactive energy management approach is proposed, which handles both temperature and lifetime at run time.

To allow integration and collaboration of all these complementary techniques, a RRM framework has been developed in [40], with the most relevant generic services.

This Chapter describes a tool flow (shown in Fig. 5.1), combining a design-time exploration with a lightweight run-time manager for embedded multi-core platforms [21]. The run-time manager leverages a set of pre-determined run-time configurations (or *operating points*) identified at design-time (see Fig. 5.1) by analyzing and exploring the architecture run-time parameters impact on the QoS through an architecture simulator. The operating points consist of information/knowledge about parameters (e.g. the power consumption, the throughput) that designers wish to optimize and resource usage associated with each configuration of the run-time parameters of the hardware/software infrastructure. The overall goal of the run-time manager is to make a reasonable assignment of the run-time parameters to optimize

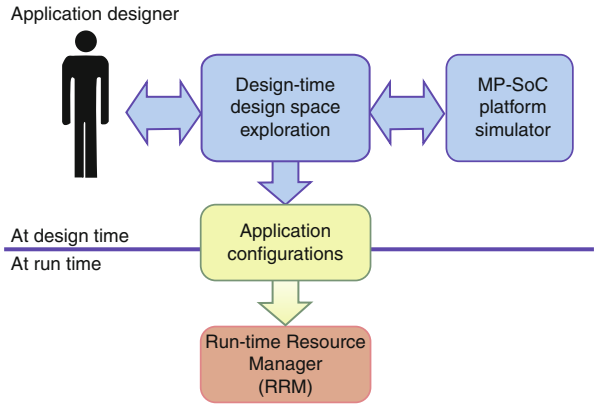

Fig. 5.1 Exploration tool-flow for Run-time Resource Management

desired output parameters (in our case execution time and power consumption) while meeting output constraints (in our case throughput QoS).

For each application, a multi-dimension Pareto set of optimal configurations is identified at design time, by analyzing and exploring several parallelizations and impacts on the constraints, the platform resource usage, and the costs. This is automated through a Design Space Exploration tool coupled with a platform simulator(s):

- The used design space explorers can be: either modeFRONTIER commercially available, or Multicube Explorer [41] (being open-source, and being the target of this Chapter). Typically in design space exploration for multi-core systems, the number of possible application configurations can be huge. Hence these exploration tools with sophisticated analytical and modeling techniques are needed within the embedded system design cycle. And as previously mentioned, they also allow alleviating the run-time decision making.
- As pointed in [13], platform simulations can be done at many abstraction levels: e.g. functional-level simulation, timed simulation, cycle-accurate simulation. But the key problem is the following one: the more accurate the simulator, the more time it takes to perform a simulation. Hence from the Design-Space Exploration point of view, which needs a large number of simulations, there is an important trade-off between the result accuracy and the simulation time. In this Chapter, used platform simulators are available at two abstraction levels [1]. In our approach, we combine two simulators during the DSE as follows: first, an extensive DSE is performed using the fast high-level timed and functional simulator HLSim [2] and derives a large set of optimal application configurations, including execution time and power consumption estimations reported by HLSim; then, only the interesting configurations are first verified and then explored further using the more accurate simulator. TLMsim is a SystemC-based cycle-accurate Transaction-Level Model (TLM) [7], built using the Synopsys virtual platform prototyping tools [33]. This simulator consumes more time, but it reports more accurate estimations. Recently, simulations based on a cycle-accurate TLM have gained importance due to standardization efforts.

In this Chapter we target both the number of cores and their frequency as a run-time configurable parameter of an application. We thus assume that:

- The task-level parallelism of each application running on the system can be changed through code *versioning*.
- The frequency associated with each core can be changed (or *scaled*) dynamically.

The overall goal of RRM is to make a reasonable assignment of the run-time parameters to reach pre-determined operating goals. Operating goals can be just one (e.g. execution time) or multiple (e.g. power consumption, throughput QoS), as in our case. In our tool flow, the RRM is incorporated on top of the basic services of the platform OS and acts as an exception handler. Its implementation conforms to the framework proposed in [40]. Its optimization strategy to globally select optimal configurations for the active applications extends the fast MMKP heuristic for multi-core run-time management [37]. In case of soft real-time applications, and when the optimization strategy can find no solution, RRM also takes the application priorities into account to relax the constraints and to reach a solution after all [21]. Also to reduce execution overhead of the run-time selection heuristic, our methodology performs initial filtering of optimal run-time parameter configurations. This allows further alleviating the run-time selection heuristic.

5.3 Run-time Resource Management Problem Definition

As described above, the goal of the proposed methodology (Fig. 5.1) for our case is to achieve, at deployment time, a desired QoS while minimizing power consumption and maximizing the usage of multiple cores. Before describing our methodology (Sect. 5.4), we formally define RRM problem. This section first describes terminologies used in this Chapter and then formally describes the RRM problem.

5.3.1 Terminologies

In this Chapter we will assume that there are p active applications; we identify each application with an identifier $\alpha \in A = \{\alpha_1 \ldots \alpha_p\}$. During its lifetime, each application can be described as having a specific **operating point** which consists of its actual **cost**, used **resources** and achieved **QoS**. A list of suitable operating points is essential for the correct behavior of the run-time manager since it represents both the *goals* to be optimized (cost, QoS and resources) as well as the independent *control parameters* (the resources and their associated properties) with which the optimization goal can be achieved.

More formally, for our optimisation problem, we can define the operating point of application α with the following tuple:

$$\boldsymbol{c}_\alpha = \langle \rho, \boldsymbol{\phi}, \pi, \tau \rangle \tag{5.1}$$

where:

- ρ is a scalar representing the **resources** associated with the operating point $\boldsymbol{c}_\alpha$. We assume that there exists an application binary version which has been parallelized over ρ cores. In our case, $\rho \in R = \{1 \ldots \rho_{max}\}$, where ρ_{max} = maximum number of available cores. Given the features of the interconnection bus, we assume homogeneity among the cores and a fixed mapping for each of the threads.
- $\boldsymbol{\phi}$ corresponds to the frequency configuration for each of the ρ cores: $\langle \phi_1 \ldots \phi_\rho \rangle$, $\phi_i \in \Phi \wedge 1 \leq i \leq \rho$.
- π is the actual **cost** associated with the current operating point $\boldsymbol{c}_\alpha$. Here, we consider the cost as the average power consumption of application α.
- τ is the average execution time needed by α for a single period[1] of an application (e.g. encoding a single frame in a multi-media application) when on the current operating point $\boldsymbol{c}_\alpha$.

In the following sections, we will use the notation $\boldsymbol{c}_\alpha[X]$ to access the element X of the tuple defined in Eq. 5.1.

5.3.2 RRM Problem

We assume that for each application α, there is an available set of operating points $\boldsymbol{C}_\alpha$ whose size is N_α. The run-time manager has to select exactly one point from each active set $\boldsymbol{C}_\alpha$, according to the available platform resources, in order to minimize the total power consumption of the platform, while respecting all application deadlines. Given a set of required application deadlines τ^α_{max}, our problem definition is to identify, at run-time, a comprehensive set of operating points:

$$\boldsymbol{\gamma} = \langle \boldsymbol{c}_{\alpha_1} \ldots \boldsymbol{c}_{\alpha_p} \rangle, \quad \boldsymbol{c}_{\alpha_j} \in \boldsymbol{C}_{\alpha_j} \wedge 1 \leq j \leq p \tag{5.2}$$

such that the following measure of power consumption is minimized:

$$\sum_{\alpha \in A} \boldsymbol{c}_\alpha[\pi] \tag{5.3}$$

[1] Note that in domains of multi-media and wireless applications, usually the applications are periodic i.e., they do same task again and again but on different input data e.g. processing video frames or wireless packets

subject to the following QoS and resource constraints:

$$\boldsymbol{c}_\alpha[\tau] \leq \tau^\alpha_{max},\ \forall \alpha \in A \tag{5.4}$$

$$\sum_{\alpha \in A} \boldsymbol{c}_\alpha[\rho] \leq \rho_{max} \tag{5.5}$$

where ρ_{max} is the maximum number of resources (or cores) in the system, p is number of active applications and A is a set of all active applications.

According to [38], the previous problem is a Multi-dimension Multiple-choice Knapsack Problem (MMKP) whose complexity resides in the NP-hard space with respect to p, N_α and ρ_{max}. Moreover, depending on how tight τ^α_{max} constraints have been set, there may not be *feasible* solutions $\boldsymbol{\gamma}$. However, in our case of embedded systems for multi-media domain, usually applications do not have *hard* real-time constraints. Instead, we address the design of a **soft real-time system** in which deadlines can be missed with the lowest penalty possible and/or the lowest probability. We manage this possibility by introducing a priority $\omega(\alpha)$ measure to be used by the run-time manager to relax some τ^α_{max} and reach a feasible solution.

In the following section we describe our proposed toolflow which consists of two parts: one design-time analysis and other run-time heuristic.

5.4 Proposed Tool-Flow for RRM

This section describes our proposed tool-flow to solve Run-time Resource Management (RRM) problem described above. Application of this tool-flow on a real-life multimedia use case is described in Chap. 9. Our tool-flow solves RRM problem in two steps:

1. A **design-time heuristic methodology** for reducing the average size N_α for each α.
2. A **run-time management layer** consisting of a filtering algorithm for each $\boldsymbol{C}_\alpha$ and a greedy, *prioritized* heuristic for solving the MMKP.

5.4.1 Design-Time Heuristic Methodology

Our design-time methodology is shown in Fig. 5.1. At design time, we identify an ordered list $\boldsymbol{C}_\alpha$ of operating points:

$$\boldsymbol{C}_\alpha = \langle \boldsymbol{c}^1_\alpha \ldots \boldsymbol{c}^{N_\alpha}_\alpha \rangle \tag{5.6}$$

$\boldsymbol{C}_\alpha$ is generated at design-time by analyzing and exploring the impact of the architecture run-time parameters on the QoS through an architecture simulator(s). Optimal $\boldsymbol{C}_\alpha$ is derived from these design-time analysis by help of sophisticated optimization

1: **for** each $\alpha \in A$ **do**
2: **for** each $\rho \in R$ **do**
3: Φ_p = arg min $\langle P_{\alpha,\rho}(\phi), T_{\alpha,\rho}(\phi)\rangle$ (ϕ is the vector of frequencies associated with the current ρ).
4: **for** each $\phi \in \Phi_p$ **do**
5: $c_\alpha = \langle \rho, \phi, P_{\alpha,\rho}(\phi), T_{\alpha,\rho}(\phi)\rangle$
6: $C_\alpha = C_\alpha \cup \{c_\alpha\}$.
7: **end for**
8: **end for**
9: **end for**
10: Pareto filter C_α on the objectives ρ, π and τ
11: Sort C_α configurations by ascending $c_\alpha[\rho]$ and (second) $c_\alpha[\pi]$.

Fig. 5.2 Design-time methodology for identifying the initial $\boldsymbol{C}_\alpha$, for each application α

and modeling techniques (see Chaps. 3 and 4) supported by DSE tool. The following algorithm describes sequence of steps taken during our design-time methodology.

Algorithm in Fig. 5.2 shows the structure of adopted design-time methodology. The algorithm mainly performs the minimization of the execution time and power consumption for each application version (i.e., for each application α and each available version of α which is parallelized over ρ resources).

Once all results are collected, these are sorted for obtaining a further speedup at run-time. A more detailed step-by-step description follows:

- *Steps 1, 2 and 3*: The algorithm loops over the set of available applications A (Step 1) and the set of resources R (Step 2) by solving a multi-objective minimization problem with respect to both $P_{\alpha,\rho}(\boldsymbol{\phi})$ and $T_{\alpha,\rho}(\boldsymbol{\phi})$ (Step 3). In our case, $P_{\alpha,\rho}(\boldsymbol{\phi})$ and $T_{\alpha,\rho}(\boldsymbol{\phi})$ are, respectively, the average power consumption and the execution time delay returned by the architectural simulator for a frequency configuration vector $\boldsymbol{\phi}$. Note that the independent variable to be optimized is $\boldsymbol{\phi}$ since ρ is set as a constraint in the loop (Step 2).
  Since the size of the design space increases rapidly with ρ, while for $\rho \leq 3$ a full search exploration is performed, for $\rho \geq 4$ the problem is solved by using the NSGA-II multi-objective genetic algorithm [10]. The genetic algorithm is run with a population size $|\Phi| \times \rho$ for 50 generations. The set Φ_p contains all the Pareto configurations associated with the solution of the problem.
- *Steps 5 and 6*: The set Φ_p is used to construct the operating points and insert them into the associated $\boldsymbol{C}_\alpha$.
- *Step 10*: It prunes $\boldsymbol{C}_\alpha$ from all the configurations that do not represent an efficient trade-off in terms of resources, energy and execution time.
- *Step 11*: It sorts the resulting $\boldsymbol{C}_\alpha$ to improve the performance of the run-time algorithm while finding an operating point with a lower resource usage while achieving τ^{α}_{max}.

The proposed heuristic methodology is able to save a considerable number of simulations. The exact amount of simulation time saving depends on the specific application scenario as well as from the selected optimization algorithm used for solving the

MOO problem in the Step 3. In particular, for the multimedia scenario proposed in Chap. 9 the design-time DSE flow is able to solve the optimization problem simulating less than the 1% of the overall run-time candidate configurations.

5.4.2 The Run-Time Management Methodology

We assume that a series of events change the application deadlines τ^{α}_{max} and trigger the RRM to identify a new $\boldsymbol{c}_{\alpha}$ for every active application α. The RRM layer is invoked as an exception handler within the OS layer and elaborates the following input information:

- Optimal application configurations $\boldsymbol{C}_{\alpha}$, for each α as determined by the design-time methodology. This information is stored within the OS layer once at the startup so there is no execution time overhead.
- QoS requirements τ^{α}_{max} for each α. In our case related to multi-media applications, this is set to average *time-per-frame*.
- A priority $\omega(\alpha)$ function which ranks each α (low $\omega(\alpha)$ corresponds to low priority).

We define, for each application α, the following input tuple:

$$\boldsymbol{\xi}_{\alpha} = \langle \boldsymbol{C}_{\alpha}, \tau^{\alpha}_{max}, \omega(\alpha) \rangle \tag{5.7}$$

The output of the run-time algorithm is a set of working operating points ($\boldsymbol{\gamma}$) achieving QoS ($\boldsymbol{c}_{\alpha}[\tau] \leq \tau^{\alpha}_{max}$) while meeting overall constraints: in our case resource ($\sum_{\alpha} \boldsymbol{c}_{\alpha}[\rho] \leq \rho_{max}$) and power consumption ($\sum_{\alpha} \boldsymbol{c}_{\alpha}[\pi]$). The operating points are then set by loading an appropriate application binary version (if the number of resources was varied) or modifying the frequency of each core as dictated by the frequency vector $\boldsymbol{c}_{\alpha}[\boldsymbol{\phi}]$.

The overall run-time management algorithm is shown in Fig. 5.3. Mainly the algorithm starts trying to find a solution which satisfies all run-time QoS constraints and then, if no solutions have been found, it iteratively relax the QoS constraint of the lowest priority application until a feasible solution is identified. In particular the algorithm consists of the following steps:

Require: run-time-manager($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
1: γ = allocate($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
2: $\Omega = \{\}$ /* relaxed application set */
3: **while** not feasible(γ) $\wedge$ ($A - \Omega \neq \emptyset$) **do**
4: $\overline{\alpha}$ = arg min $\omega(\alpha)$, $\alpha \in (A - \Omega)$
5: $\rho_{\overline{\alpha}}$ = min $c[\rho]$, $c \in C_{\overline{\alpha}}$
6: $\tau^{\overline{\alpha}}_{max}$ = min $c[\tau]$, $c \in C_{\overline{\alpha}} \wedge c[\rho] = \rho_{\overline{\alpha}}$
7: γ = allocate($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
8: $\Omega = \Omega + \{\overline{\alpha}\}$
9: **end while**
10: **return** γ

Fig. 5.3 Priority based QoS-aware run-time management

- *Step 1*: The algorithm invokes an *allocate* function which solves the actual MMKP problem with the given constraints to produce a solution $\boldsymbol{\gamma}$.
- *Step 2 and 3*: The invocation is performed iteratively until $\boldsymbol{\gamma}$ satisfies the given constraints on the resources or all the application constraints have been relaxed ($\Omega = A$).
- *Steps 4, 5 and 6*: Whenever the solution is unfeasible in terms of overall resources, the deadline τ^{α}_{max} of the lowest priority application $\overline{\alpha}$ is relaxed (Step 6) reducing it to the minimum possible among a reduced set of resources $\rho_{\overline{\alpha}}$.
- *Step 7*: *allocate()* algorithm is invoked.
- *Step 8*: The selected $\overline{\alpha}$ is then put into a *relaxed application set* Ω.

The *allocate* function (see Fig. 5.4) actually solves the MMKP problem with a light-weight algorithm as presented in [38].

- *Step 2*: $\boldsymbol{C}_{\alpha}$ is pruned in order to have a single $\boldsymbol{\phi}$ assignment for each ρ. This is done by selecting only those configurations which meet τ_{max} but have the lowest power consumption for a unique ρ. The resulting set of operating points is put in $\boldsymbol{\chi}_{\alpha}$ and actually reduces the amount of data to be elaborated by the MMKP solver considerably. Note that the pre-filtering Step 2 has a linear complexity with respect to the cardinality of $\boldsymbol{C}_{\alpha}$ due to the sorting performed in Fig. 5.2, Step 11.
- *Step 4*: A *unified knapsack* vector $\boldsymbol{\Psi}$ is created; this is a vector of $\boldsymbol{c}$ sorted by prioritizing the *improvement* per single resource (*normalized user value*) $u(\boldsymbol{c}_{\alpha})$:

$$u(\boldsymbol{c}_{\alpha}) = \frac{v(\boldsymbol{c}_{\alpha})}{\boldsymbol{c}_{\alpha}[\rho]},\ v(\boldsymbol{c}_{\alpha}) = \max_{\forall \boldsymbol{c} \in \boldsymbol{\chi}_{\alpha}} \boldsymbol{c}[\pi] - \boldsymbol{c}_{\alpha}[\pi] \tag{5.8}$$

  In other words, $v(\boldsymbol{c}_{\alpha})$ is the improvement in terms of power consumption with respect to the maximum power consuming operating point in $\boldsymbol{\chi}_{\alpha}$. $u(\boldsymbol{c}_{\alpha})$ is $v(\boldsymbol{c}_{\alpha})$ per unitary resource.
- *Step 5*: A greedy linear-scan algorithm [38] is invoked on $\boldsymbol{\Psi}$ for identifying the final set of operating points $\boldsymbol{\gamma}$. Given the sorting performed previously, the algorithm has a worst case complexity of $O(pN\log(pN))$, where N is the average size of the operating point set per application.

It can be shown that the total worst-case complexity of our RRM scheme is $O(pN_{max} + pN\log(pN))$, where p is number of active applications, N is a set of all active applications and N_{max} is maximum number of configurations. This reduction in complexity is achieved due to the adequate filtering and sorting done before the greedy algorithm solving the knapsack problem. Running such an algorithm on a

Require: allocate($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
1: **for** each α **do**
2: $\quad \chi_{\alpha}$ = pre-filter($C_{\alpha}, \tau^{\alpha}_{max}$)
3: **end for**
4: Ψ = merge-and-sort ($\chi_{\alpha_1} \cdots \chi_{\alpha_p}$)
5: γ = solve-knapsack(Ψ)
6: **return** γ

Fig. 5.4 Allocate function

host processor will not cause too much overhead in terms of resources and execution time (Chap. 9).

5.5 Conclusions

In this Chapter, we presented an automated tool flow which tackles Run-time Resource Management challenge for multi-core embedded systems by efficiently combining a design space exploration tool coupled with platform simulator(s). This tool flow consisted of two steps: first a design-time heuristic methodology was described which reduces number of configurations that an application can be run optimally. This first design-time phase leverages optimization algorithms described in Chap. 3 for minimizing the amount of simulations to be performed. Second step is a light-weight run-time manager which selects an optimal configuration for each running application depending on demanded QoS requirement. Hence, at run-time, run-time manager leverages the design-time DSE results for deciding an operating configuration to be loaded for each application. This operation is performed dynamically, by following the QoS requirements of the specific use-case. Application of this tool-flow on a real-life multimedia use case is described in Chap. 9 while other frameworks operating at the Operating System (OS) level are presented in Chap. 6.

References

1. Avasare, P., Vanmeerbeeck, G., Kavka, C., Mariani, G.: Practical approach for design space explorations using simulators at multiple abstraction levels. In: Design Automation Conference (DAC) Users' Track (2010)
2. Baert, R., Brockmeyer, E., Wuytack, S., Ashby, T.: Exploring parallelizations of application for mpsoc platforms using mpa. In: Proceedings of IEEE Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1148–1153. France (2009)
3. Baker, T.P.: An analysis of edf schedulability on a multiprocessor. IEEE Transactions on Parallel and Distributed Systems **16**, 760–768 (2005)
4. Ballesi, P., Fornaciari, W., Siorpaes, D.: A hierarchical distributed control for power and performances optimization of embedded systems. In: Conference on Architecture of Computing Systems (ARCS), pp. 37–48. Hannover (2010)
5. Benini, L., Bogliolo, R., De Micheli, G.: A survey of design techniques for system-level dynamic power management. IEEE Transactions on VLSI Systems **8**, 299–316 (2000)
6. Buchard, A.: Assigning real-time tasks to homogeneous multiprocessor systems. Technical Report, University of Virginia (1994)
7. Cai, L., Gajski, D.: Transaction level modeling: an overview. In: Proceedings of CODES+ISSS (2003)
8. Chan, H.L.: Non-migratory online deadline scheduling on multiprocessors. In: Proceedings of SODA, pp. 970–979 (2004)
9. Chen, J.J., Yang, C.Y., Kuo, T.W., Shih, C.: Energy-efficient real-time task scheduling in multiprocessor dvs systems. In: Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific, pp. 342–349 (2007)

10. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. Proceedings of the Parallel Problem Solving from Nature VI Conference pp. 849–858 (2000). URL citeseer.ist.psu.edu/article/deb00fast.html
11. Garey, M.R., Johnson, D.S.: Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman (1979)
12. Gheorghita, S., Palkovic, M., Hamers, J., Vandecappelle, A., Mamagkakis, S., Basten, T., Eeckhout, L.: System-scenario-based design of dynamic embedded systems. ACM Trans. on Design Automation of Electronic Systems **14**, 1–45 (2009)
13. Gries, M.: Methods of evaluating and covering the design space during early design development. Integration, the VLSI journal **38**(2), 131–183 (2004)
14. Isci, C., Buyuktosunoglu, A., Cher, C., Bose, P., Martonosi, M.: An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 347–358 (2006)
15. Lauzac, s.: Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In: Proceedings of EUROMICRO, pp. 188–195 (1998)
16. Li, J., Marinez, J.: Power-performance implications of thread-level parallelism on chip multiprocessors. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, pp. 124–134 (2005)
17. Li, J., Marinez, J.: Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In: Proceedings of the International Symposium on High-Performance Computer Architecture, pp. 77–87 (2006)
18. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM **20**(1), 46–61 (1973)
19. Luo, J., Jha, N.K.: Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In: Proceedings of IEEE ASP DAC, pp. 719–726 (2002)
20. Marchal, P., Jayapala, M., Souza, S., Yang, P., Catthoor, F., Deconinck, G.: Matador: an exploration environment for system design. Journal of Circuits, Systems, and Computers **11**(5), 503–535 (2002)
21. Mariani, G., Avasare, P., Vanmeerbeeck, G., Ykman-Couvreur, C., Palermo, G., Silvano, C., Zaccaria, V.: An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In: DATE 2010 - International Conference on Design, Automation and Test in Europe., pp. 196 –201. Dresden, Germany (2010)
22. Mariani, G., Palermo, G., Silvano, C., Zaccaria, V.: A design space exploration methodology supporting run-time resource management for multi-processor systems-on-chip. In: Proceedings of the IEEE Symposium on Application Specific Processors, pp. 21–28. San Fransisco, USA (2009)
23. Miniskar, N., Hammari, E., Munaga, S., Mamagkakis, S., Kjeldsberg, P., Catthoor, F.: Scenario based mapping of dynamic applications on mpsoc : A 3d graphics case study. In: Proceedings of SAMOS Workshop, pp. 48–57 (2009)
24. Miniskar, N., Munaga, S., Wuyts, R., Catthoor, F.: Pareto based run-time manager for overlapped resource sharing. In: Proceedings of the ECES Workshop. Belgium (2009)
25. modeFRONTIER DSE tool, http://www.esteco.com
26. Munaga, S., Catthoor, F.: Proactive reliability-aware energy management in hard real-time systems - a motivational case study. In: Proceedings of the Workshop on Design for Reliability, pp. 195–200. Cyprus (2009)
27. Prasanna, V.K., Yu, Y.: Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In: Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on, pp. 341–348 (2002)
28. Ramamritham, K., Fohler, G., Adan, J.M.: Issues in the static allocation and scheduling of complex periodic tasks. IEEE Real-Time Systems Newsletter **9**, 11–16 (1993)
29. Schmitz, M., Al Hasimi, B., Eles, P.: Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems. In: Proceedings of IEEE Design, Automation Test in Europe Conference Exhibition (DATE), pp. 514–521 (2002)

30. Shin, D., Kim, J.: Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In: Proceedings of ACM International Symposium on Low Power Electronics and Design, pp. 408–413 (2003)
31. Shojaei, H., Ghamarian, A., Basten, T., Geilen, M., Stuijk, S., Hoes, R.: A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management. In: Design Automation Conference (DAC), pp. 917–922 (2009)
32. Sinha, A., Chandrakasan, A.: Jouletrack - a web based tool for software energy profiling. In: Design Automation Conference (DAC), pp. 220–225 (2001)
33. Synopsys virtual platform prototyping tools. http://www.synopsys.com
34. Yang, P., Marchal, P., Wong, C., Himpe, S., Catthoor, F., David, P., Vounckx, J., Lauwereins, R.: Multiprocessor Systems-on-Chip: Cost-Efficient Mapping of Dynamic Concurrent Tasks in Embedded Real-Time Multimedia Systems, Eds W. Wolf and A. Jerraya. Morgan-Kaufman (2004)
35. Yang, P., Wong, C., Marchal, P., Catthoor, F., Desmet, D., Verkest, D., Lauwereins, R.: Energy-aware runtime scheduling for embedded multiprocessor socs. IEEE Design and Test of Computers **18**(5), 46–58 (2001)
36. Ykman-Couvreur, C., Catthoor, F., Vounckx, J., Folens, A., Louagie, F.: Energy-aware task scheduling applied to a real-time multimedia application on an xscale board. Journal of Low Power Electronics **1**(3), 226–237 (2005)
37. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management. In: Proceedings of the International Symposium on System-on-Chip, pp. 195–198. Tampere, Finland (2006)
38. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In: Proceedings of International Symposium on System-on-Chip, pp. 1–4 (2006).
39. Ykman-Couvreur, C., Nollet, V., Marescaux, T., Brockmeyer, E., Catthoor, F., Corporaal, H.: Design-time application mapping and platform exploration for mp-soc customized run-time management. IET Computers and Digital Techniques **1**(2), 120–128 (2007)
40. Ykman-Couvreur, C., Obermaisser, R., El Salloum, C., Goedecke, M., Zafalon, R., Benini, L.: Resource management for embedded multi-core platforms. In: Proceedings of DATE Workshop on Designing for Embedded Parallel Computing Platforms. France (2009)
41. Zaccaria, V., Palermo, G., Mariani, G.: Multicube explorer (2008). http://www.multicube.eu
42. Zhang, Y., Hu, X., Chen, D.: Task scheduling and voltage selection for energy minimization. In: Design Automation Conference (DAC), pp. 183–188 (2002)

Chapter 6
Run-Time Resource Management at the Operating System Level

Patrick Bellasi, Simone Corbetta, and William Fornaciari

Abstract Current hardware platforms provide the applications with an extended set of physical resources, as well as a well defined set of power and performance optimization mechanisms (i.e., hardware control knobs). The software stack, meanwhile, is responsible of taking direct advantage of these resources, in order to meet application functional and non-functional requirements. The support from the Operating System (OS) is of utmost importance, since it gives opportunity to optimize the system as a whole.

Purpose of this chapter is to introduce the reader to the challenge of managing physical and logical resources in complex multi- and many-core architectures, focusing on emerging MPSoC platforms.

6.1 Introduction

Modern applications are of a wide variety, and they all have different requirements in terms of hardware and software resources, performance goals, power and energy constraints. What we are experiencing in today's electronics is a continuous convergence of different application classes into the same environment: life-critical, real-time, mobile and general processor power are converging to the same System-on-Chip (refer to Fig. 6.1).

The increasing demand for applications and scenarios translates into an increasing demand for processing power and integration, as it can be seen from Fig. 6.2. This adds several challenges to the design of such applications. In addition, to cope with design and implementation costs, a suitable design methodology should contemplate the re-use of part of the subsystems. In this context, the Resource Manager (RM) should be able to operate on slightly different applications, with different resources and with as little as possible porting efforts.

P. Bellasi (✉)
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
e-mail: bellasi@elet.polimi.it

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_6, © Springer Science+Business Media, LLC 2011

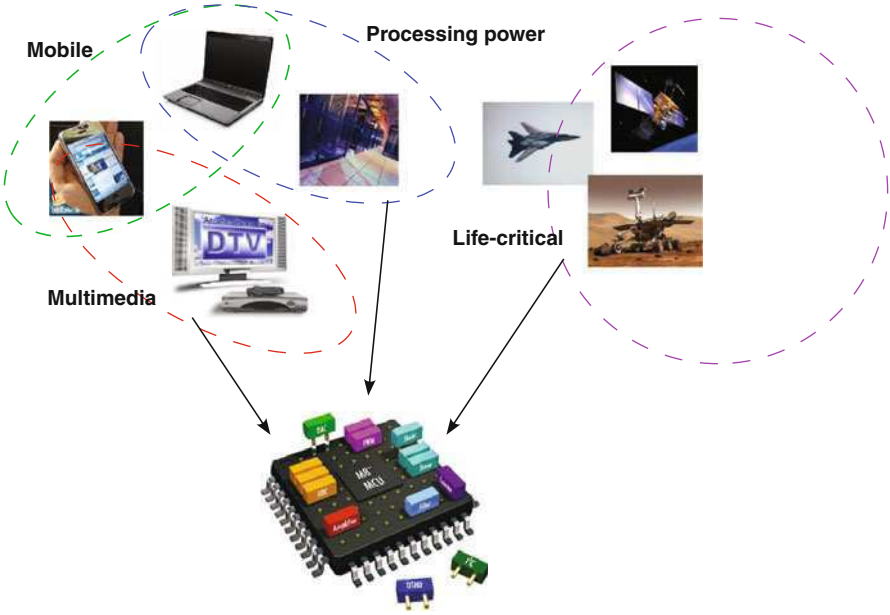

Fig. 6.1 Different applications scenarios are converging to the same SoC design paradigm

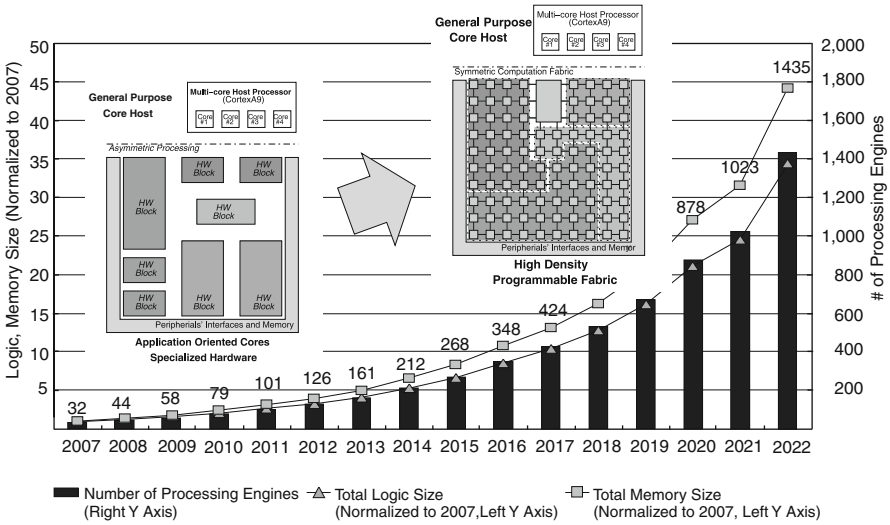
General Purpose Core Host
Multi–core Host Processor (CortexA9)
Asymmetric Processing
HW Block
Peripherials' Interfaces and Memory
Application Oriented Cores Specialized Hardware
Symmetric Computation Fabric
High Density Programmable Fabric
Logic, Memory Size (Normalized to 2007)
# of Processing Engines
2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
32 44 58 79 101 126 161 212 268 348 424 878 1023 1435
Number of Processing Engines (Right Y Axis)
Total Logic Size (Normalized to 2007,Left Y Axis)
Total Memory Size (Normalized to 2007, Left Y Axis)

Fig. 6.2 Increasing computing power density brings to ever more complex architectures with an increasing count of per-chip functionality and resulting in more complex applications. The chart represents the Processing Elements (PE) density per technology year [8]

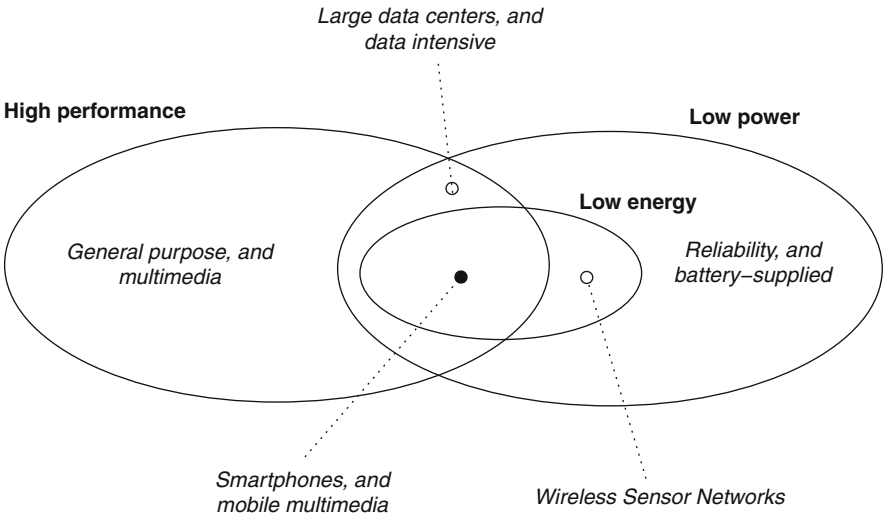

Fig. 6.3 Application domains are mapped to their respective design requirements. According to the required power/performance ratio, we can detect several application domains, in both the mobile and non-mobile case

Without any loss in generality, we can point out two main broad classes: *high performance* and *low power*. High performance encloses all applications for which high operating frequency and high throughput are experienced, while low-power applications are those for which energy budgeting is of major importance. Notice that speaking of "power" or "energy" is not the same, and the use of one or other term relies on the target application scenario. Refer to Sect. 6.4 for a more detailed discussion on this distinction. In conjunction, technology and design advances lead to a proliferation of mobile embedded systems, for which several additional challenges raise up. The combination of the aforementioned requirements and the application platforms (i.e., mobile against non-mobile), gives us free room for a simple yet comprehensive classification of the interested scenarios in the MPSoC domain. This fact is graphically shown in Fig. 6.3.

Five slightly different domains have been detected. General high-performance applications comprise both mobile and non-mobile domains. In the non-mobile case, for instance, we can find desktop personal computers, where the user experience from the performance view-point is of utmost importance; along with those requirements, such applications are mainly general purpose, meaning that they are designed for a wide spectrum of jobs, without precise optimization of power or energy aspects. Nevertheless, it is true that there exist cases in which high performance and low power are a common goal, as in systems collecting sensible data, such as large data centers; in this case the peak power has to be minimized, in order to reduce cooling costs, increase system reliability, and decrease device failure rate. General low-power operations are required by almost all mobile embedded systems, but different approaches should be taken while considering high performance and energy

savings domains. Mobile multimedia devices (e.g. Smartphones) fall in this category. The effective trade-off between very high performance, for instance to ensure audio and video capabilities, and ultra low-power operation is a challenging task. Last, applications for which energy savings are of primarily relevance but in which performance can be lower than in any other case exist: wireless nodes in a Wireless Sensor Network belong to this set.

The remainder of the chapter is organized as follows: Section 6.2 gives an insight of the problem of managing resources at run-time, while discussion on the support from the OS is given in Sect. 6.3. A more detailed review of the existing Power Management (PM) frameworks based on the Linux kernel is presented in Sect. 6.4. Conclusions are given in Sect. 6.6.

6.2 Run-Time Resource Management (RTRM)

So far we have seen how modern MPSoC architectures provide an exceptional flexibility, with a huge set of physical and logical resources. The presence of multiple applications that can be potentially integrated within the same chip makes the problem of managing such resources a challenging problem. By "Run-Time Resource Management" (briefly, RTRM) we hereby mean the set of processes, techniques, methodologies and instruments that allow to *use* the available resources, provided an objective function. Generally speaking, resource utilization is subject to specific objectives and constraints, depending on both the effective hardware implementation and the considered application. One of the objectives of the Resource Manager is thus to ensure that all such constraints and requirements are met, while satisfying the Quality-of-Service (QoS). A RTRM will perform this action dynamically, because there will be no chance to know which will be the applications (and, as a consequence, the requested resources) a priori.

Purpose of this section is to introduce in a more formal way the concepts related to the problem of managing the resources, the run-time component and the role of the Run-Time Manager.

6.2.1 Problem Overview

The concept of *resource* is at the same time enough general to hide some details on its real nature (e.g., physical or logical implementation) and enough detailed to allow us to employ it in a constrained optimization problem. Thus, a generic resource can either refer to a processor, a memory subsystem in a memory hierarchy or it can also refer to the computation time fraction assigned to a task. Either ways, each resource has a precise meaning within its own context. The set of available resources in a system can be defined as the set $R = \{r_1, r_2, ..., r_N\}$ of all those N resources we have access to. Each resource r_j in R can be of any kind, and we

can further assume that there exist several *types*, that are clearly dependent on the context. $C = \{c_1, c_2, ..., c_M\}$ is the set of M types (or classes) of resources. A static mapping from the resources to the corresponding classes exist, and it can be meant as being (quasi) independent on the target application. Such mapping $M \subseteq R \times C$ is defined by an ad-hoc function $map(r_i, c_j)$, taking as inputs a generic resource $r_i \in R$ and a generic class $c_j \in C$. At any instant of time, only a subset $AR \subset R$ of the resources is available, representing the resources that are not in use by any other task in the system. Complementary, the set $BR = AR^c$ keeps track of the used (busy) resources. Considering the set $T = \{t_1, t_2, ..., t_P\}$ of P tasks in the system, there will exist a mapping from the system resources to the tasks: $U = \left\{u \in R \times T \;\; and \;\; u = uses\left(r_i, c_j\right), \;\; with \;\; r_i \in R, c_j \in C\right\}$. The obvious relation $U \perp R = BR$ holds, where the $\perp$ operator is the projection operator, returning the set of resources from a collection of pair-like sets.

The choice U of how to map resources to tasks, i.e. the implementation of the *uses* function depends upon a specific objective function. Such objective function gives priorities to specific resources for requesting tasks, and such priority changes according to the objective function. Generally, the objective function can be modeled through a set of metrics: performance, memory throughput, execution latency, power consumption, and so on. Thus, roughly speaking, the role of the Resource Manager is to fill in the set U with the appropriate entries, according to the *current* implementation of the *uses* function. Such implementation can change at run-time, for instance due to changing workloads or application requirements. Hence, there is a need for a Run-Time Resource Manager component, that can cope with the changing application scenario, resources availability and requirements.

6.2.2 Resource Manager Overview

Not every RTRM is suitable for each application/platform configuration. Several differences are experienced passing from one application to one other, and so the mapping would change. For this reason, a minimal set of requirements has to be guaranteed from the RTRM view-point. The following presents a subset of such requirements;

1. **Flexibility**, RTRM should be able to work with the expected specifications under different scenarios. Flexibility ensures the reuse of the overall methodology, impacting positively on the design and implementation costs of the entire project. In addition, flexibility leads multiple applications to efficiently use the resources, without incurring in unwanted overheads;
2. **Scalability**, RTRM should work well for an increasing number of cores and, in general, of active resources. This is dictated by the growing and broad convergence toward multi- and many-core architectures. Scalability is hereby intended from two view-points: performance (latency, overhead) and energy efficiency (it should not consume more power than required, otherwise benefits will be outperformed);

3. **System-wideness**, decisions made by the manager should not interfere with other decisions previously taken, i.e. assigning specific resources to a newly incoming task should not impact negatively on the already running tasks. In order to ensure this requirement, the RM should have system-wide perspective of what's going on in the system at any instant of time.

The previous list shows which are the main requirements for a Resource Manager to be suitable for the job. The effective implementation of the RM depends on several requirements related to the hardware and software architectures. Before giving an overview of the main components defining the manager, we define which are the classes of resource managers that can be found. This is just an adoption of the work done in [16]. The various classes of resource management can be identified according to the following metrics: static versus dynamic, hardware versus software implementation, centralized versus distributed, adaptive versus static.

Static resource managers are somehow hard-coded in the system software, in the sense that the decisions taken are defined in advance, according to a predefined policy. Policies cannot be changed at run-time so no flexibility is achieved in this case. In contrast, *dynamic* approaches let the managing process evolve according to the needs. The policies as well as other useful parameters might be changed during execution time, either explicitly (i.e., user-driven) or implicitly (i.e., guided by the current scenario).

The implementation can be either hardware, software or a smart mix of the two. Purely *software* approaches are very flexible, but they incur in high overhead; purely *hardware* approaches, on the other hand, provide high performance at the cost of flexibility. A mixed *hardware/software* co-design can optimize the performance and the flexibility. The real challenge consists of finding which part will be in hardware and which part will be in software. Examples of purely software or hardware approaches are reported in [16].

Centralized approaches provide a single entity that gathers all the necessary information from the underlying hardware and from the upper software. *Distributed* approaches, instead, provide multiple control points, that gather local information. The former technique lacks of scalability as well as efficiency, since with a huge amount of resources to handle, the number of information to be gathered would increase exponentially. On the other hand, distributed approaches have only local view of the problem, and thus do not provide a system-wide optimization point-of-view. For these reasons, a more suitable approach should have a *hierarchical* control, i.e. integrating both local- and system-wide views of the same problem.

Last, decisions should be taken with respect to an adaptive or non-adaptive mechanism. *Adaptive* mechanism can follow the workload changes in the system, and they are clearly dynamic in nature. In contrast, *non-adaptive* (also called *fixed*) approaches are of a static nature and do not adapt their behavior to the changing context.

The selection of the implementation details of the aforementioned design parameters affects how the manager will behave in the target system. The most interesting approaches reported in [16] can thus be classified as shown in Fig. 6.4.

OMAP platform from Texas Instruments [4] provides a software and centralized/distributed implementation, in which a single master Resource Manager resides

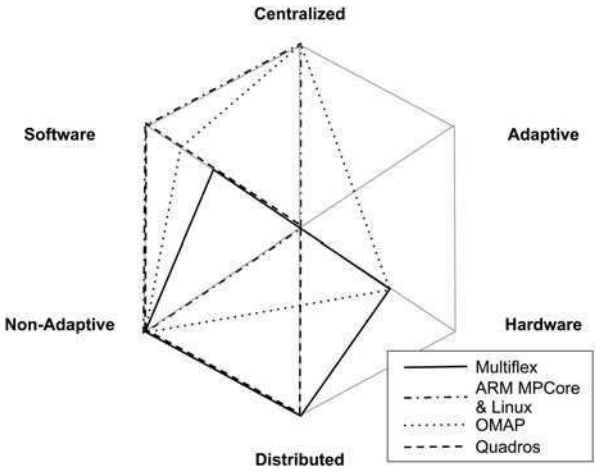

Fig. 6.4 Different commercial approaches are reported comparing different features for a RTRM. Data is taken from [16]. For each interesting feature, a value of 0 stands for no support, a value of 1 stands for full support, while a value in between (e.g., 0.5) means the feature is only partially supported. This is especially useful while comparing hardware and software implementations

on the central general-purpose processor (the "host"), and a slave RM server resides on each system co-processor, e.g. DSP. The role of the master RM is to manage and allocate slave DSPs, create tasks and allocate communication resources.

The approach followed by RTXC RTOS is based on the replication of RTOS kernel services on each core of the target platform containing MPSoC and multiple DSP coprocessors. However, there is no actual RM as we might think, since it is in charge of the designer to decide for the allocation of tasks. This happens through the use of the RTLib, but no run-time adaptation is provided [16].

The ARM MPCore is a multiprocessor platform for general-purpose operating systems like Linux. In this context, Linux would act in SMP fashion, hiding the fact that multiple Processing Elements (PEs) are present. In this way, the RM is embedded in the underlying OS, and transparently provides to the applications a multi-core environment with several available resources to speed-up application execution.

Last, Multiflex from STMicroelectronics addresses Nomadik platforms [21] containing multiple general purpose processors executing either Linux or Symbian in SMP configuration, and several customized application processors (ASIP) for video, audio and 3D processing. Resource management is performed through a master-slave configuration, like the one presented for OMAP platforms. Coprocessors are effectively seen as *devices* where to push tasks for execution.

6.2.3 Run-Time Manager Components

This section presents a bird's-eye-view on the main components that a Resource Manager is made by. This section provides a well-defined set of the components *specifying* an RM, yet from a high-level perspective. This has been inspired by the work in [16], presenting a valid, but yet incomplete, modeling of the RM components and subsystems.

The core of an RTRM consists of the subsystem taking decisions about the resource-to-task allocation problem; in other words, the core of an RTRM

consists of the implementation of the *uses* function. Such implementation resides in a separate component, that takes as input (also) the asserted constraints on the platform. Those constraints come from the software applications, and represent the "contract" between the software and the hardware. The effective allocation of the resources happens according also to the (current) objective function in the system. The objective function, as we said in Sect. 6.2.1, defines which are the candidate resources for the requesting application. A *QoS Estimator* component is employed for the sole purpose of estimating the potential resource allocation, so that to compare it with the application requirements. If an appropriate resource-to-task mapping is found, then the allocation is given back to the application, that can now begin the desired execution. The concept of "resource configuration" conveys in a single entity any kind of resources: physical resources (e.g., assigned cores), logical resources (e.g., clock cycles) and non-functional aspects (e.g., voltage/frequency selection for power-constrained systems). For this reason, we may want to call such configuration an *Operating Point*. In this perspective, the selection is performed by an *Operating Point Selection* module. Such module implements the core of the RM. It is based on two entities: *mechanisms* and *policies*. Mechanisms represent the observation and control knobs that the underlying hardware (or the HAL software) provides for the desired purpose, while policies define *how* resources to be allocated are selected.

From the description above, it is clear how an RTRM stays in the middle: it takes software-related and application-specific constraints and retrieves information from the underlying hardware platform. An overview of the RTRM in a system-wide perspective is shown in Fig. 6.5. Three different aspects are then taken into

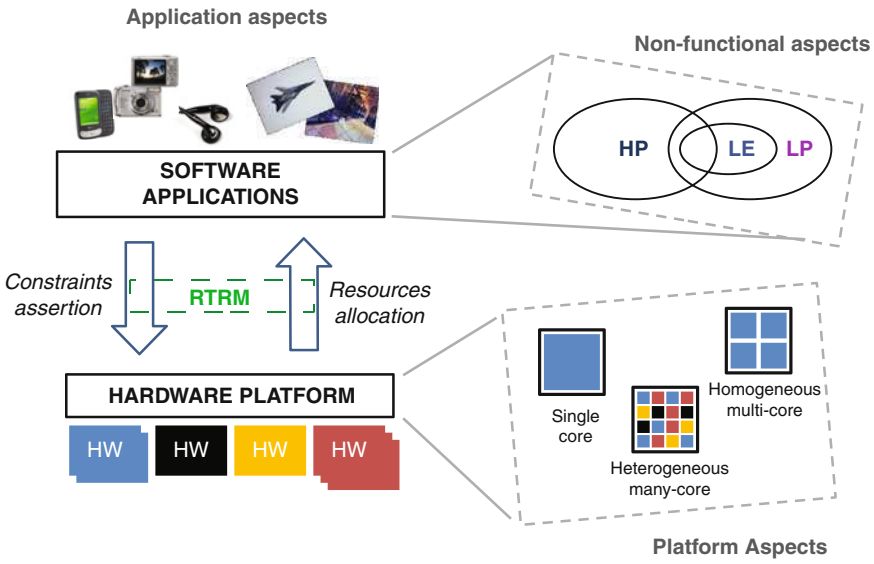

Fig. 6.5 The RTRM takes into consideration application aspects, non-functional aspects and platform aspects

account: *application*, *non-functional* and *platform* aspects. Application aspects refer to application requirements and constraints. Non-functional aspects collect those QoS-related, such as power consumption, throughput, memory bandwidth; these can be used directly to setup the objective functions. Last, platform aspects come from the hardware: available resources, hardware architecture features, and so on.

6.3 Operating System Support

In single-user single-tasking systems, when all the available resources were under direct control of the user, resource control was straightforward. Instead, with the advent of multi-tasking systems, the need for more complex resource control mechanisms raised. In these systems, one of the main goals is to control resources by sharing them fairly among all concurrent tasks. This requires the implementation of suitable "accounting mechanisms" and a "scheduling algorithm". Accounting mechanisms are used to keep track of the available resources and their allocation to requesting clients. Instead, scheduling algorithms should support the contrasting requirements of fairness and granted access to resources to all the clients.

This problem becomes particularly complex if we consider systems with a mix of best-effort and critical workloads. Regardless of their nature, critical workloads are characterized by the requirements of a certain number of resources to grant their functionalities. Conversely, best-effort workloads could tread performance for resources without compromising functionality. In this scenario, a properly designed run-time resource manager should be aware of these differences and provide an adequate support for priority access to available resources.

To tackle the resource management problem, different architectural alternatives have been evaluated. Both middleware and native operating system support have been developed and quantitatively evaluated, especially on the specific context of multimedia applications [23]. In the rest of this section we focus on the native operating system approach, highlighting its role on the resolution of the resource management problem. Then we focus on the Linux kernel, which is a widely adopted operating system for a broad range of application contexts, and we describe the main mechanisms it offers to manage resources at run-time.

6.3.1 System-Wideness and the OS

The operating system support for resource management has changed significantly in the course of the last decades. One can find in existing literature many studies on resource control at operating system level started. Systems were mainly based on mainframes, which have a central computation system and many users accessing it remotely and competing to the usage of really limited resources. Thus, the basic approach of dividing the available resources into several groups corresponding to

different user profiles, e.g., gold, silver and iron, was pretty enough. For example, resources were always granted to "gold users" and temporary given to "silver users" once not needed by the first class. This simple approach allows to maximize the usage of the limited resources without any adverse impact on the expected performance of different users.

Nowadays we have multi-processor systems characterized by multiple CPUs and high bandwidth network (that too getting faster and faster) and with cheaper memories. These systems have been adopted initially on data centers, which use clusters of computation nodes, and the focus has moved on security and web based distributed access. This application scenario raised the interest for the virtualization support. Indeed, the business idea is to partition the resources of a big system (i.e., guest system) into smaller pieces (i.e., host systems), each one dedicated to different customers which actually pay for what they get in terms of resources. Thus, the new requirement has become the possibility to run in isolation multiple virtual systems on the same hardware resources.

The requirement to control the resources of a big system, in order to partition them in smaller pieces, has been supported by three different design approaches (Fig. 6.6), which rely on: virtual machine, OS containers or flexible resource control. The *virtual machine* approach allows to run even a completely different OS on the same hardware platform [15]. While this approach grants the maximum security [27], thanks to the complete isolation of the host systems, it is also the one with higher maintenance efforts and run-time overheads [9]. Moreover, running hosts on complete isolation reduces also the run-time flexibility since it is practically impossible to share unused resources with overloaded hosts. The *container* based is a better approach for the performance and maintenance perspective. In this case the isolation is provided by a single OS that has complete control over all the available resources and gives each host system a partitioned view of them [12]. The Resource Control provided by a single OS allows better performance and a more flexible run-time resource allocation, thus maximizing their usage.

Being mainly addressed to resource isolation and security, these two approaches are interesting for application contexts in which we are interested into running different customers on the same hardware resources. This is the main case of web services,

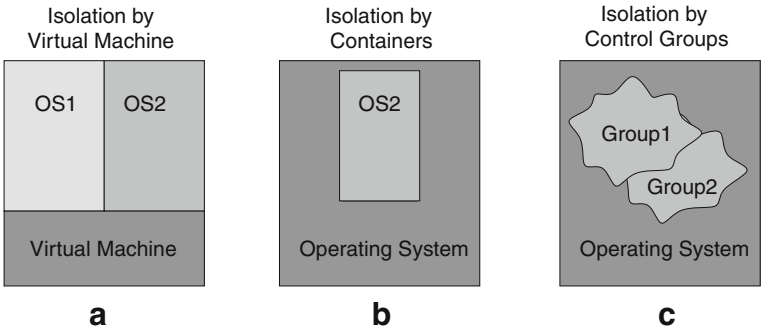

Fig. 6.6 The three different possible levels of resource control inside an OS

Table 6.1 The main properties of three different design approaches for resource management

	Virtual machine	Container	RC
Performance	Not good	Very good	Good
Isolation/Security	Very good	Good	Not good
Runtime flexibility	Not good	Good	Very good
Maintenance	Not good	Good	Good

where multiple customers, even belonging to different companies, buy the usage of portions of the same physical resource. Of course this is not the only application scenario. In many other scenarios, we are interested in optimizing the usage of available resources by many tasks of the same system. This is the main case of mobile embedded devices, such as smartphones and set-top-boxes. In these application contexts we have a single user which runs multiple tasks corresponding to different workloads, either critical or best-effort. In these scenarios, the entities are tasks, instead of users, which compete to the usage of shared resources, instead of buying them. The resource management problem, in this specific scenario, can be efficiently solved by using more flexible resource control systems. The latter approach is still based on a Run-Time Manager running on a single OS which groups the available resources and then maps tasks on these groups according to some usage policy.

The main properties of the three different approaches are summarized in Table 6.1. In the next subsection we concentrate on the third approach to investigate deeply the support that a modern Linux kernel provides for the flexible control of resources.

6.3.2 OS Mechanisms Supporting RTRM

Almost all modern OSs provide resource management mechanisms to properly grant shared resources access to multiple tasks. Linux is perhaps the OS that has developed the most advanced mechanisms to support run-time resource management for a large number of resources. This is thanks to its wide adoption, on systems that range from high-end servers in data centers down to multimedia mobile systems in smartphones. As shown in Fig. 6.7 many Linux subsystems provide a Resource Manager, most notably: CPU, memory and the I/O communication channels such as disks and networks interfaces. Moreover, since power and energy consumption are critical aspects, for both high-end servers and mobile devices, the power management subsystem works parallel to the previous ones to manage properly electrical consumptions.

Past proposals on resource management and task grouping were based on a basic abstraction that allows to group together multiple processes in order to track and limit the resources that they are allowed to access. To fulfill all these functionalities, the Task Control Groups (TCG) framework is available since Linux kernel version 2.6.24, when it was first developed and merged by Paul Menage of Google Inc.

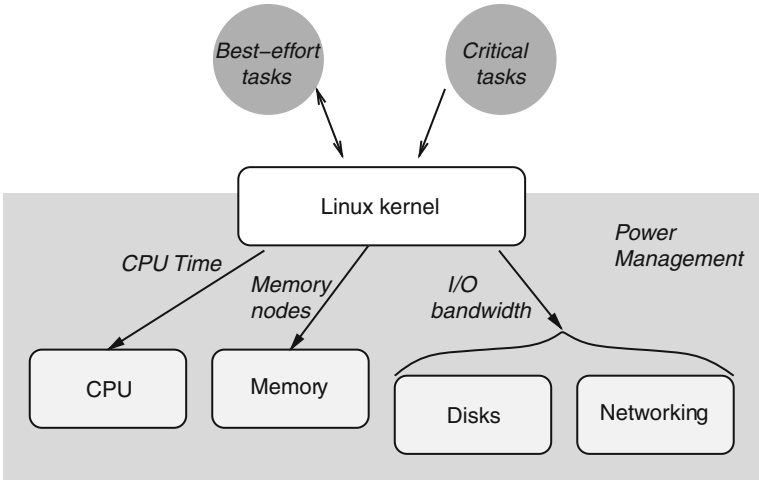

Fig. 6.7 Linux provides RTRM support for different subsystems. Tasks can be grouped in classes which correspond to different resources access priorities

When a task is associated to a particular *cgroup* (i.e., a control group), it will get an access to a portion of the system resources, where we can specify how big or small that share can be. These shares represent minimum values, not maximum. Thus, if we give one group 10% of a resource and another 90%, then if the more privileged group isn't using its full 90%, the other group can have whatever is left over. This borrowing mechanism allows to optimize resource usage while still granting each group the resources defined at design time whenever required.

This low-level mechanism was added in Linux mainly for containers and virtual machines. However, they are not restricted for that solely purpose. One of the most interesting approaches nowadays is to employ it to manage resources—and therefore performance—of ordinary processes in a multi-tasking single user system. The idea is that other subsystems hook into the generic cgroup support to provide new attributes for them. For example, this allows to account or limit the resources that the tasks in a control group could access.

Like many other Linux frameworks, the TCG interface is organized as a virtual file-system, which can be used both to inquire about resources partitioning and to configure it by simply reading and writing files.

6.3.2.1 CPU Time Management

The CPU subsystem allows to manage the CPU time resource. Different cgroups can be created and assigned to. The client of this subsystem is the Linux scheduler, which recalculates the percentage of the total CPU each cgroup will get. For example, we might create a cgroup for the background processes, another for the logged-in users

and a third for root and daemons. If we gave them each one share, they would get a guarantee of no less than 33% of the CPU time.

6.3.2.2 Memory Management

The next precious resource that can be managed is memory. The `memory` container allows to put a memory limit on a cgroup, and if the group's Resident Set Size (RSS) exceeds that number, its least-recently-used pages are swapped out.

This is definitely excellent for managing tasks with memory leaks, thus improving system stability. Moreover this can also be used for less severe problems. For example, tasks which use a lot of memory tend to push out pages of other tasks which haven't run lately, making them slow in restart. By putting a memory limit on the cgroup of memory, hungry tasks will grant a more rapid restart of other tasks.

Unfortunately, at least at the moment of this writing, the cgroups memory limit is a hard limit, and doesn't allow overcommitment even when we have lots of free memory. Thus, a natural extension to this container is the support for a soft limit, which should guarantee an upper bound like CPU share does. Such a support should allows a larger soft limit when there is plenty of free memory, but it reduces the limit to the smaller hard ones when there is demand for memory.

6.3.2.3 I/O Management

A certain kind of abrupt slowdown is common to anything that can build up a queue, such as a CPU, a disk or a network, as well as any program that uses these resources. The queue network theory [13] shows that any open system with a service center s that requires certain service time t_s to process a job exhibits performance which has the behavior represented in Fig. 6.8. This is the classic behavior of a disk, where for example it can deliver 425 I/Os per second at 100% utilization. Thus, since we

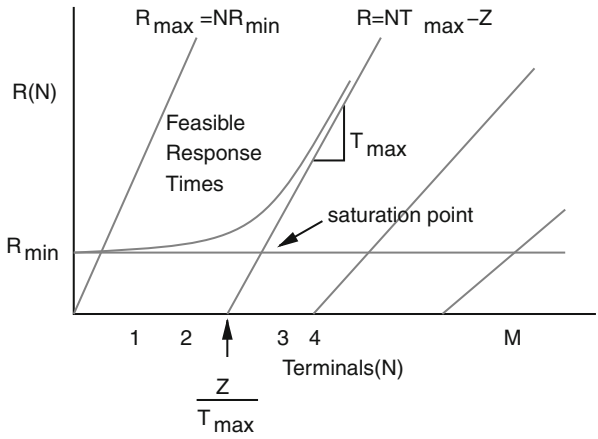

Fig. 6.8 The asymptotic system response time of an open system

can't exceed 100% utilization, if we ask for 500 I/Os per second, only 425 are served while 75 disk access requests have to sit in a queue and wait.

To keep a devices in the "good" part of their response curves is an excellent reason to use resource limits, thus avoiding slowdowns. Four hundred requests at 40 ms is far better than 500 averaging more than 80 ms each. Some experiments could give a rough estimate of the limit of a device, and we can use this value to set a limit that keeps them from being driven into overload. The solution to this problem is completed by limiting the I/Os that a task is allowed to issue. Thanks to its I/O scheduler, which is the natural client of the block subsystem, Linux shines on the management of resource such as disks and other input/output devices. The I/O scheduler has been extended to understand cgroups, and can refrain from dispatching I/Os if they will exceed the cgroup bandwidth ration. As usual, using the virtual file-system interface, the `blockio` subsystem allows to define a [MB/s] bandwidth limit of each char or block device node, by simply echoing that value within the provided `blockio.bandwidth` attribute.

6.3.2.4 Network Management

The control of the network bandwidth resource is straightforward by using the cgroup framework, but slightly different with respect to the previous subsystems. As in case of previous subsystems, a specific subsystem is provided for the resources, i.e. `tc`, which can be used to define classes of network traffic. However, the control groups defined are used to associate a "class id" to every network packet generated by tasks belonging to the corresponding group. The actual bandwidth control is in charge of the Linux traffic shaping framework, which is the client for this control group subsystem. A command line tool allows to associate bandwidth constraints to each class id defined by control groups.

6.3.2.5 General Conclusion on Resource Management

The first observation is about usability of the task control group framework: resource sharing is not easy to reason about. Task control group framework only guarantees that when the machine is loaded you get a specific resources share. However, when the system is not loaded a task could get more, but how much is probabilistic. It depends on everything else that is running on the machine, and at the end this can be confusing and also hard to handle for some tasks.

Because of the hard understanding of their behaviors, one could be tempted to only use hard limits. Hard limits are trivial to reason about, and they are genuinely useful for preventing catastrophes, such as a disk driven into infinite slowdown. However, hard limits should not be used as a general tool: doing so means that the tasks can never use any spare cycles that a system has. Instead any such spare cycles will be wasted, which is the same as wasting resources.

6.4 Power Management

Digital electronics gives enough opportunities to reduce power consumption at different abstraction levels, not only through silicon physical optimization [24, 25]. As a matter of fact, power reduction opportunity increases with higher levels of abstractions, so that from architectural up to system software layer we have enough room to address the power/performance challenging trade-offs.

The involved abstraction levels are shown in Fig. 6.9. We have to decouple the SoC design in two planes: the software plane and the hardware plane. The former, shown in Fig. 6.9a, relates to the different levels at which the software can operate to effectively give contribution to power consumption reduction. The latter (Fig. 6.9b), on the other hand, refers to the design of the underlying hardware, providing mechanisms to the upper levels of details.

From a software perspective, power reduction techniques can be employed both statically and dynamically. Static strategies are generally addressed at compile-time [7, 22], or at least through ad-hoc software architecture techniques at source-code level. Static techniques are of great importance since they can be used to exploit as much as possible the required power/performance requirements, but these techniques lack of flexibility. This is much more true in those systems where the workload is not known in advance. For this scenario, dynamic approaches should be employed, for instance at the OS level (kernel) or at higher abstraction levels [2]. Applications can directly impact on the power/performance trade-off, but a more sophisticated mechanism can reside at the kernel level, where the OS is aware of the entire system status. The most complete software frameworks for the Linux kernel are reviewed in the next sections.

From a lower level perspective, hardware has to provide the software with control and observation points in order to ensure that the desired goal is achieved. Thus, the power/performance trade-off solution is searched in a hardware/software co-design approach, as it has been previously stressed in Sect. 6.2. While conventional

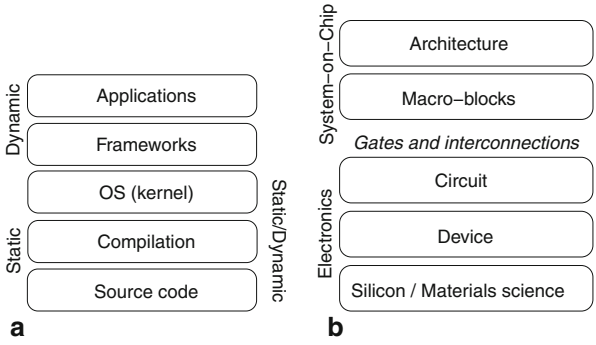

Fig. 6.9 Power reduction and optimization techniques cover a wide range of SoC design, from both software and hardware planes. A holistic low-power design methodology, where applicable, should consider crossing different abstraction levels for efficient and proficient power management and optimization. **a** Software design and abstraction levels, **b** Hardware design and abstraction levels

Table 6.2 Summary comparison among the presented software frameworks. The classification is based on static or dynamic power consumption, clock gating, Multiple Voltage Scaling and power gating

		Power optimization		Clock gating	MVS	Power gating
		Static	Dynamic			
Pure OS	`CPUFreq`		•		•	
	`CPUIdle`	•		•	•	•
	`S/R Fw`	•				•
	`Clock Fw`		•	•		
	`V/I Fw`		•		•	•
Cross-Layer	Centralized (`DPM`)	•	•	•	•	•
	Distributed (`QoS`)	•	•			
	Hierarchical (`CPM`)	•	•			

low-power design methodologies define mechanisms to solve power issues from the physical up to the gate and architectural levels of abstraction, such methodologies are generally based on a precise hardware support, e.g. level shifters, PLL registers for clock signal [10]. In parallel, there are several software frameworks that address power management. Hereby, we will focus on those designed for Linux-based systems, and which were originally designed for general purpose platforms. Nevertheless, their applicability is of (quite) general validity, also for mobile embedded systems.

The available approaches can be conveniently grouped into two categories: pure-OS and cross-layer. The distinction comes from the power optimization mechanisms that are applied, and which kind of interaction is exposed to higher levels. A summary of the presented approaches is given in Table 6.2.

The table reports the proposed classification in terms of pure-OS and cross-layer, and for each entry a comparison is performed against the power optimization and power optimization mechanisms involved: static versus dynamic, and clock gating versus power gating or voltage selection. A bullet (•) suggests that the current entry addresses that specific optimization, or supports that specific mechanism. Table 6.2 considers only three mechanisms for power management. Clock gating technique aims at reducing the dynamic power consumption by disabling (i.e., gating) the input clock signal. The frequency of the clock signal drops to zero, so that the switching activity drops to zero, too. Multiple Voltage Scalings (MVS) refers to the use of different power rails, providing different regions of the chip with an ad-hoc voltage supply value. Last, power gating is the technique that cuts input voltage source, so that reducing quadratically the consumption of dynamic power.

6.4.1 Pure-OS Techniques

Pure-OS techniques are completely implemented at the Operating System level; they do not provide support for direct input from applications. They attempt to figure out application requirements based on previously monitored behavior or current activity, and enforce some control decision either on a single device or on an entire subsystem. We can further divide these techniques in two groups, whether they tend to

optimize static or dynamic power consumption. In the former case, namely *resource hibernation*, they are generally based on the exploitation of ON/OFF states of the peripherals. In the second case we refer to *resource tuning* techniques, since power minimization is obtained by properly configuring available operational parameters of the target platform, according to the changing run-time requirements. We have to further distinguish between device-specific techniques and system-wide techniques. The former class relates to those techniques addressing specific devices, while the latter attempts to optimize the system as a whole, in a more abstract view of the application.

6.4.1.1 Device Specific Techniques

Being one of the more power demanding device, power optimization of the system processor is of major interest. There are two main frameworks available in a modern Linux kernel: one is devoted to reduction of static power consumption while the other addresses the optimization of dynamic power consumption.

CPUIdle

The `CPUidle` framework [18] focuses on power management of an idle CPU. We refer to a CPU as being *idle* when it is doing nothing useful for the application itself, there is no workload, and hence it can be turned off to prevent from unnecessary power consumption. We have several opportunities in this context, ranging from clock gating or shutting down increasing portions of the circuitry, down to completely power gating the processor. These different solutions correspond to a well-defined set of idle states that modern high-performance processors exhibit. Idle states are characterized by particular processor configurations, with precise power consumption levels and wake-up latency. Moving from the simple approach of clock gating to power gating, there are increasing penalties, mostly related to wake-up latency. For instance, waking-up from an idle state requires just to re-enable the clock, while waking-up from a deep idle state could require to re-initialize the CPU and restore its registers from main memory too.

`CPUidle` addresses power/performance trade-offs from a software layer standpoint, with the aim of exploiting all the available idle states of a processor without impacting on the overall system performance. An effective solution to this problem requires an adequate support to identify the real system requirements in terms of CPU latency. The current Linux implementation defines a proper software design which separates the low-level software mechanism from high level interface toward the framework clients to simplify this. An overall view of the framework architecture is given in Fig. 6.10.

The low-level interface [19] supports the definition and registration of processor-specific drivers. Those drivers are required to define the set of idle states available on the target CPU. Each state must be characterized by a set of attributes defining their

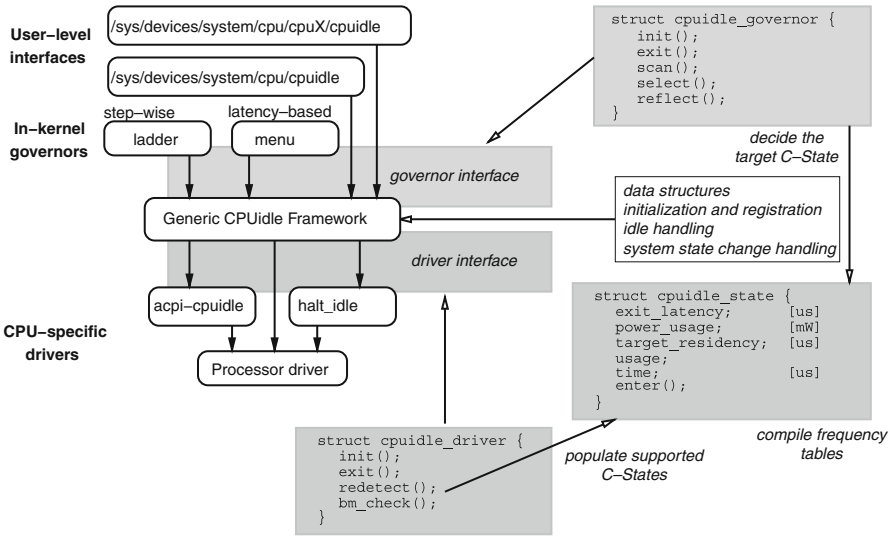

Fig. 6.10 A simple representation of the `CPUidle` framework

power contribution, exit latency and a target residency time which is considered as necessary to get an advantage from entering that state. Every idle state could also be associated to a specific callback function which implements all the required low-level code needed to actually enter the state.

The high-level interface provides support for the definition of a *governor*, a processor-independent algorithm for choosing the effective idle state to enter, according to system constraints on maximum latency. There might be more than one governors registered in the core, but just one can be used at any time. Widely used implementations provide two governors, called *ladder* and *menu*. The ladder governor adopts a step-wide policy: every time the CPU is idle, a deeper idle state is entered only if we were previously able to remain in that state for a period greater than its corresponding target residency. Instead of relying on a simple heuristic approach, the policy implemented by the menu governor is latency-based. This policy exploits the information on the maximum allowed system latency in order to identify the idle state that should be reached every time there is an opportunity. This governor is certainly more efficient but requires a closer collaboration among applications and kernel drivers, to collect such requirements.

The core implementation is completely platform independent and provides the glue code that defines the required data structures, support drivers, governors registration and run-time selection. A proper monitoring interface is also exported to the collecting statistics on idle states usage.

CPUFreq

`CPUfreq` [20] focuses on the optimization of dynamic power consumption by exploiting DVFS mechanisms. A processor is in an *active state* when there is some

workload ready to be executed. A workload can either be CPU-bounded or I/O-bounded; the former requires intensive CPU computations on memory located data, while the latter presents a more heavy information exchange toward relatively slow peripherals such as disks or low-bandwidth buses. In general, a single task cannot be exclusively classified in a single class; it happens that some portions are more CPU-intensive, while others are more like I/O operations. This means that the nature of a task could change during its execution; the combination of different workloads is even more evident if we consider a multi-tasking system with many concurrent applications running at the same time and sharing the few available processors.

The CPUfreq framework considers these combined behaviors in order to optimize power against performance. The basic idea is to exploit the possibility to perform computations at different operative frequencies. The set of available frequencies define the *performance states* of the platform; lower frequencies correspond to lower voltages and thus also less performing states with reduced power consumption and increased execution time. Switching from one performance state to an another one inserts an overhead that must be kept into consideration. Moreover, there is a need to identify efficiently the real system performance requirements. These observations make the CPU frequency scaling a rather complex mechanism to exploit. The framework available in Linux simplifies the implementation by a proper software design which aims at decoupling low-level software mechanisms from high level policies. An overall view of the software architecture is depicted in Fig. 6.11.

The low-level software mechanisms [11] are implemented by *drivers*, required to define both platform specific information, and a set of control routines. The required information is related to the available performance state and the corresponding transition overheads, while the platform specific hardware mechanism to actually perform

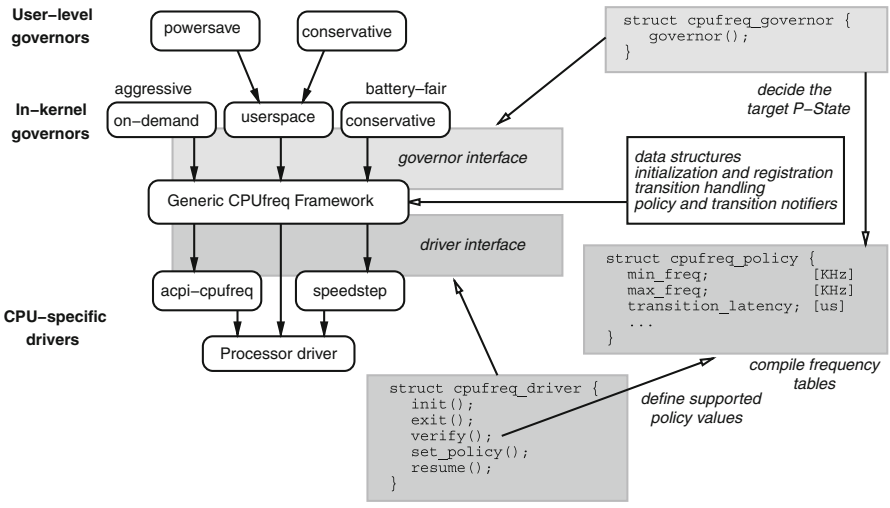

Fig. 6.11 A simple representation of the `CPUfreq` framework software design

a transition must be wrapped by a set of properly defined callback functions. An high-level interface allows to define a *governor*, which is the platform independent algorithm for the evaluation of system performance requirements and of the selection of the optimal performance state. At least one governor must be defined, and multiple governors enable adaptive and dynamic multiple optimization strategies. The default framework implementation provides five governors, the more interesting and widely used being the *on-demand governor*. It implements a scaling policy based on the *run-to-idle* optimization. The CPU load is monitored in a periodic time frame, and according to the load observed in the past time frame a scaling decision is taken according to a simple rule: try to keep the CPU utilization around the 80% [20]. On CPU utilization higher that that threshold, an immediate scaling up to the maximum available frequency is required, to the contrary, on lower CPU utilization the scaling down is required step-by-step but only after a preconfigured number of negligible load time-frames are elapsed.

The *core implementation* provides the code to bind the platform independent governors down to the architecture specific driver. Moreover, a proper notification API is provided which allows other kernel components not only to be aware about CPU scaling operations but also to somehow interact with those optimization decisions, for example to assert a *veto* on some changes due to some contingent constraints.

6.4.1.2 System-Wide Techniques

The clock distribution tree and the power domains have some common characteristics: they have system-wide view (i.e., they interact with all the available on-chip devices) and they define a hierarchical dependency tree (i.e., a local power optimization decision could impact on different devices). These two components require system-wide optimization techniques which are able to collect information from multiple devices in order to identify a proper optimization strategy.

Suspend/Resume

The *Suspend/Resume Framework* (`S/R`) provides the proper support for a complete and efficient resource hibernation strategy. Linux supports three static-power saving states: *standby*, *suspend-to-RAM*, and *hibernation*. The main difference between them stands on how the device state is preserved. In a standby state a device is not functional, but it is still powered at least to grant the preservation of the content of its configuration registers. This kind of power saving addresses static power optimization, since the device logic is powered down and only a retention voltage is applied to the configuration array. This state could be always entered whenever a device is not in use since the recovery time is relatively short and practically negligible if compared to the typical operating system reaction time. In suspend-to-RAM a device is completely powered off, the contents of the configuration array are moved backed up in a secure area in main memory. Recovering from such a state is more time demanding

since all the peripheral configurations must be recovered from main memory, and sometime this is possible only after a proper cold-start device initialization procedure. *Hibernation* is the more effective saving state: power consumption minimization is at its optimal value, saving the system configuration in a persistent storage and powering off all devices (memory included in some cases). Unfortunately, as one can argue, this last state is also the most expensive in terms of recovery time. A complete system restart is generally required, and it is done during the boot-up procedure in order to keep dynamic overhead at a minimum.

The main challenge for a successful implementation is the proper tracking of device functional dependencies. Different devices in a system could be interconnected to form subsystems. For instance a USB device, such as a memory stick, is connected to the port of an HUB which in turn connects to a port of a USB host controller. All this chain define a USB subsystem. Finally the host controlled could be either a system device or a gateway towards a PCI bus; which in turn defines another subsystem. Considering all the devices within a system and their inter-dependencies with respect to their functional dependencies what we get is logical dependency tree rooted at the CPU and having a device at each end node. This tree specifies an implicit order that must be respected both on suspend, starting the suspension from nodes and visiting the tree up to the root, and on resume, by converse visiting it starting from the root node down to the device nodes.

Clock Framework

The `Clock Framework` has been introduced in the Linux kernel to optimize the dynamic power consumption associated to the clock distribution tree. The proposed framework is based on the management of the system clock signal. The hierarchical generation and distribution of the clock signal opens several opportunities for engineers to reduce power. The effective validity of the approach is driven by the fan-out value and the switching activity of the clock signal.

Purpose of the framework is to export the programmability of such components to the software level [26]. In this way, it is possible to cut-off some tree edges according to the desired computational activity; this is actively done by switching off a selected subset of LDO, PLLs or DIV modules. The approach takes even more advantage in some partitioned systems, in which several independent subsystems receive the clock from a common source, the top-level system clock, and scale the input signal according to local optimization policies, using DIV modules. In this way, individual operating requirements can be locally addressed with little silicon cost.

There are two main mechanisms for clock management: *clock stopping* and *clock scaling*. The former technique allows to disconnect the clock line from the associated PLL, and to eventually power off the PLL. Such mechanism gates the clock to the entire sub-tree controlled by the actual PLL that has been turned off. Clock scaling, on the other hand, does not disable clocks, but it instead scales down the incoming signal using physical dividers or reprogramming the top PLL for the current sub-tree.

Voltage and Current Control Framework

The *Voltage and Current Control Framework* (`V/I Framework`) [5] is a quite specific support focusing on the efficiency of voltage regulators. In the architecture exploration, we highlighted how modern SoC architectures are composed of multiple voltage domains to fit specific requirements of each hardware block. In general, the voltage domains within a SoC could have some dependency relation between them. Sometimes these voltage domains are directly controlled by a dedicated voltage regulator usually provided by an external companion chip.

Each device in the system is powered by a certain voltage domain and, according to the specific functionalities required by a device, the current drained from the domain could also be very different. For instance, if we consider an audio-codec controller, its current drain is very different if we are listening to some audio stream via a loudspeaker or we are simply performing some digital audio mixing activities. A physics study of the dynamics of a regulator device shows that its efficiency is highly affected by the instantaneous current load. The *Regulator Power Efficiency (RPE)* of a regulator is defined in Eq. 6.1.

$$RPE = P_{out} / P_{in} \tag{6.1}$$

Equation 6.1 compares the amount of power P_{in} that is presented as input to the regulator, and how much P_{out} we are able to derive from it; it is a direct measure of how much energy is lost in the regulator itself.

It is known that when the regulator works in normal mode, it is able to efficiently support only current loads over a certain threshold value. On the contrary, once the current load on the corresponding voltage domain drops under this threshold, the current requirement could be satisfied with a better efficiency only switching the regulator to an idle operating mode. This kind of behavior of voltage regulators are worth to be considered in order to implement a really holistic approach to power management in a modern embedded system. The framework presented in this paragraph has been introduced in the Linux framework quite recently, but provides a well designed and mature support to simplify the exploitation of this kind of optimization. The framework is composed of four separate interfaces:

- *regulator*, allows a regulator driver to register a set of required operations to the core framework;
- *consumer*, allows a device to notify voltage and current requirements to the regulator driver;
- *platform*, allows the system platform code to define the voltage domains, their dependencies and thus the creation of the regulator tree;
- *userspace,* exports a lot of useful voltage/current data and operation mode statistics via a `sysfs` interface to support device power consumption and status monitoring.

6.4.2 Cross-Layer Techniques

Mechanisms and techniques supporting power management can be implemented at different abstraction levels; not only at architectural level but also at software level. Applications are aware of their Quality-of-Service (QoS) expectations. For instance, if we consider the playback of a network video stream: then we could easily identify at the application level some of the requirements, e.g., in terms of network bandwidth and decoding processing workload. Thus, the development of holistic approaches should support the aggregation of data from multiple layers into power management decisions. Cross-Layer techniques try to exploit mechanisms from different abstraction levels at the same time. The idea behind them is to provide properly defined mechanisms to collect abstract information from the higher abstraction levels, i.e., user-space applications, and exploit them to give some useful hints to the lower abstraction level techniques in order to improve the exploitation of the available architectural mechanisms. Cross-layer techniques aggregate data from multiple layers into power management decisions; indeed a properly defined interface allows the user space to assert Quality-of-Service requirements and exploit these information to support system-wide optimization. These techniques could be further grouped into two categories, *centralized* and *distributed*. *Centralized techniques* have been developed mainly to support the power optimization of relatively simple and dedicated embedded systems (e.g. personal media player), but these have some scalability problems related to their complexity which impact on the implementation effort. On the other hand, *distributed techniques* are designed to be more scalable to easily address much more complex architectures (e.g. new generation smart-phones).

The power optimization techniques proposed in this class are essentially based on the definition of a single coordination entity, which stands in between the user-space applications and the available architectural mechanisms. However, we could identify essentially two orthogonal approaches: centralized and distributed; the main difference is in the role of the coordination entity. In centralized approaches the coordination entity has a direct control on the available mechanisms which are used to perform power management according to a single and system-wide optimization policy driven by the requirements collected from user-space. Distributed approaches, instead, not only implement a lightweight, single, and system-wide optimization policy but also exploit many other devices and subsystem specific policies. The idea is to implement a distributed control model where user-space requirements are aggregated and used to feed some input to more specialized local controls.

6.4.2.1 Dynamic Power Management

The `Dynamic Power Management` software (and hardware, refer to architecture, presented in [3]) is both an architectural and interface proposal for a centralized cross-layer technique targeting high-performance embedded systems. Purpose of this proposal is to exploit effective power management mechanisms from the architectural view-point and from the management view-point at the OS level. This framework

is neither a DVFS algorithm, nor a power-aware OS and even nor a mechanism such as ACPI. Its relevance comes from the integrated engineering that has been applied to provide a highly efficient power management solution. To this purpose, the framework architecture is based on few abstraction objects: operating points, task states and policies. Each one cooperates information for performance and power management purposes. An overall representation of these abstractions is depicted in Figure 6.12.

An *Operating Point (OP)* is the lowest level abstraction which encapsulates a mixture of physical and logical parameters, representing a power-related sensible characterization. Each OP is thus a specific set of $\langle parameter, value \rangle$ pairs corresponding to a precise system power/performance configuration. At any given instant of time, the system is allowed to execute in a specific OP. Examples of operating points for a processor [17], as specified for the PowerPC architecture, are: core voltage, CPU operating frequency, bus frequency, and memory timing. The designer is in charge of the choice and setting of the OPs, as many as required by the capabilities of the target platform and the desired complexity of the framework implementation. The framework allows also the definition of *congruence classes (CCs)* which are sets of OPs that could be considered to be equivalent from certain power/performance optimization strategy. A *task state (TS)* is the high-level abstraction corresponding to a possible system operating state. In the control model defined by DPM, the system is seen as a state machine defined on a limited and well defined set of states. Example of states could be: idle, interrupt handling, CPU-bound process, I/O-bound process. The definition of the actual set of TS is once again in charge of the integration engineer. At run-time, each task could be associated with a task state. This mapping allows to identify in which task state the system is by simply looking at what task is scheduled to run at every time instant. Thus, switching from one task to another could imply the switching of the system among different task states.

Since each task state might have its own power/performance profiles, it is worth defining a mechanism to map task states to operating points, or more in general to a congruence class. This is achieved through the introduction of the *policy* abstraction, representing such mapping. According to the time of running task, the DPM core

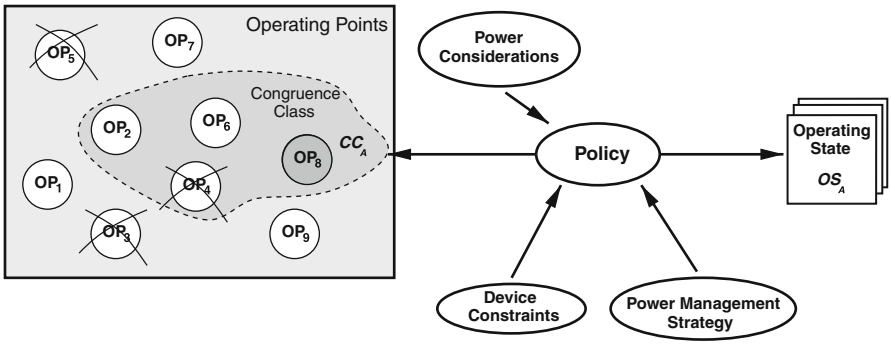

Fig. 6.12 The DPM architecture abstraction objects

framework is able to automatically identify the current task state and accordingly map this on a congruence class defining a limited set of eligible operating points. Identifying a congruence class is not sufficient to actually select the best OP. To that purpose two more concepts are considered in the framework: the constraints and the optimization strategy. A *constraint* is a requirement on a specific OP value that could be asserted by either applications or device drivers. The core framework collects constraints asserted by all system entities and use them to invalidate the OPs that are not compatible with them. This first mechanism could thus reduce the number of eligible OPs available in the current congruence class. Finally, where more OPs that are still valid after considering all the constraints, the *optimization strategy* defines the ultimate rules to give each valid OP a relative preference value.

The framework is mostly an architectural proposal which requires customization efforts for each specific platform in order to be effectively used. The core framework provides just the *glue* code with the basic mechanisms to define the abstraction objects, but their actual definition is entirely an effort of the platform engineer. Secondly, the definition of the abstraction objects is a rather complex problem by itself since it requires a deep knowledge of a platform as a whole. Nevertheless, this remains one of the more interesting proposal for a centralized cross-layer optimization framework which is worth to consider especially in the case of relatively simple and dedicated embedded systems that require fine grained and low-overhead control.

6.4.2.2 QoS-PM Framework

The `QoS-PM Framework` has been the first attempt to implement a sufficiently generic framework to support distributed cross-layer power management within the Linux kernel. This kernel infrastructure has been proposed by Intel, essentially as an extension to the pre-existing *Latency Framework*, for optimizing the power consumption of a WiFi network interface [6].

The basic idea of this framework is to define a set of QoS parameters which are available to both applications and in-kernel code to assert requirements on them. The parameters defined are sufficiently abstract not to reduce the portability of the solution; in the current implementation they represent network throughput, network time-out and system latency. Of course this initial set of parameters is quite limited, but could be easily extended provided that the new parameters are completely platform independent. Well-defined and simple methods can be used to assert requirements on each of these parameters which are then aggregated by the core framework. The *requirements aggregation* is performed using a simple boundary function, i.e., the maximum or the minimum of the requests is considered to be the more restrictive value for the parameters. Drivers and other kernel code could declare their interest on a particular parameter by simply subscribing the corresponding notification chain. Once a new request on a parameter happens, the aggregated value is notified by the core framework to each driver or subsystem which has registered to the notification chain associated to that parameter.

Once a driver is notified by a new aggregated value for a certain parameter of interest, it could exploit that information in order to fine tune its local optimization policy. For instance, in the current implementation of the CPUidle framework described so far, when an idle state transition has to be decided, it takes into consideration the system latency requirements. This information is valuable since we know that the exit time from an idle state could be highly varying and thus could have also a great impact on the experienced latency of the system. The behavior implemented in this framework is thus that of a distributed control model. The framework core collects requests from applications and provides a simple optimization policy, based on the boundary aggregation, that deliver some tuning parameters to many other specialized policies. It is worth noticing that the current implementation of the notification mechanism support only a *best-effort* approach. Once a driver receives a notification of an update on a certain parameter, it could decide to do its best to satisfy the requirements. But if that is not possible, there is no feedback delivered up to the requesting user-space application.

The best-effort nature of the current implementation, along with the simple aggregation functions supported, are justified by the need to keep the QoS framework as simple as possible. By this design choice, the paradigm of a cross-layer distributed approach to power management is easily exported to the Linux kernel. Nevertheless these are also some of the main limitations of the current implementation and motivate the research interest in this specific area of power management at operating system level.

6.4.2.3 Constrained Power Management

The `CPM framework` [1] is the first complete and comprehensive implementation of the *hierarchical power manager* concept for the Linux kernel. This framework is based on the design of a single coordination entity which allows both to exploit a system-wide view of resources availability and to aggregate all the application requirements.

Resource availability is defined by device drivers, in a platform independent way, and this information is exploited by the framework to support the system-wide optimization policy with fine-details and improved portability. The fine-details are granted by the low-level information collected at run-time by drivers, thus improving the portability of the control solution. By changing an architecture or even a single device, the new information is automatically detected.

Application requirements are collected by a single and well defined user-space interface. The framework aggregates all the requirements and translates them in a set of constraints for the global policy.

At run-time, a global optimization policy could exploit all the information collected either by drivers and applications: the former defining resource availability while the latter asserting QoS requirements. This information could be used effectively to solve a multi-objective optimization problem targeted to identify the best system-wide configuration. For scalability reasons, this configuration could not be

completely defined by the coordination entity. Instead, this entity will notify proper constraints to drivers and let their local optimization policies to do the fine-tuning.

In order to efficiently support portability and scalability of the control policy, it has been defined on the base of abstraction and modeling layers, as depicted in Fig. 6.13.

The *abstraction layer* grants portability without compromising the fine-details requirement. Available resources and devices working modes are represented in a platform independent way. Resources are abstracted using a set of metrics (PSM/ASM) which can be used also to setup the multi-objective optimization

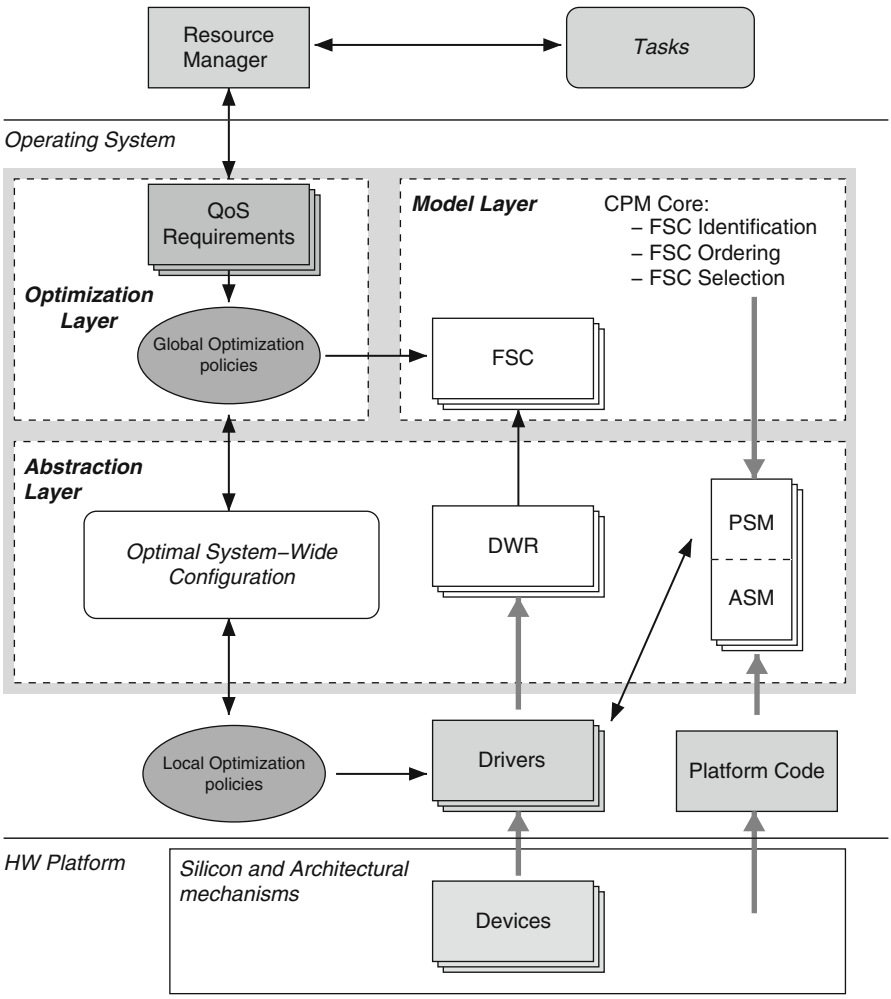

Fig. 6.13 The `CPM framework` architecture is based on three layers: abstraction, modeling and optimization

problem. A working mode of a device can also be represented in the space of these metrics by identifying a corresponding device working region (DWR).

The *model layer* exploit these abstraction information to automatically build a representation of all the system-wide feasible configurations (FSCs), each one identifying a working points of the entire system where a certain QoS level can be granted.

This model is suitable for supporting an efficient global optimization strategy provided by the *optimization layer*. This policy could support a multi-objective optimization strategy defined on the considered metrics.

The framework provides the implementation of a global optimization policy which rely on Linear Programming to identify a solution-equivalent and efficient optimization strategy. This strategy is based on three main tasks: FSC Identification, FSC Ordering and FSC Selection.

The *FSC Identification* exploits the information provided by the abstraction layer. This layer provides a multi-dimensional solution space, defined by the optimization metrics (PSM/ASM), and the device working regions (DWR) defined by each device. These information are used in the model layer to automatically identify the set of Feasible System-Wide Configurations (FSC) that identifies a platform independent representation of the system resources and capabilities.

In the *FSC Ordering* task, the global optimization policy exploits the FSC-based abstract system view provided by the model layer, to order the FSC previously identified according to the multi-objective optimization goal. Each time the system use-case change, the optimization goal is updated, and thus the FSC should be re-ordered.

Application requirements are collected and translated into constraints which invalidate unfeasible FSCs. The optimal FSC is selected, considering both the previously identified ordering and their validity defined by the current constraints, by running the *FSC Selection* task.

Each task has different run-time overhead and activation frequency. The FSC identification is the more complex task, it is required just at system-boot. Instead, FSC selection must run each time an application assert a requirement but it has a negligible impact thanks to the support provided by the previous tasks.

6.5 Exploiting DSE to Support RTRM

The Design Space Exploration (DSE) techniques have been demonstrated to be a valuable tool for the exploration and optimization of hardware platforms. However, their usage at run-time for the optimization of software behaviors has been only recently explored [14]. Even if the approach is promising, the integration of a DSE based optimization policy for the power optimization of a computing platform could exhibit less flexibilities. This is especially true if we consider the new generation systems, based on a deep sub-micron production processes, which run multiple applications on top of an Operating System.

This section highlight the potential shortcomings and outline a possible strategy for the integration of the DSE results with the Operating System and its Run-Time Resource and Power Management frameworks. At the time of this writing, this integration will be a part of future investigations, hence it is outside the scope of this book on MULTICUBE project.

6.5.1 Integration Pitfalls

There are two main obstacles to exploit DSE results at system-level: the possibility to have a mixed workload and the run-time phenomena. They are discussed below.

6.5.1.1 Mixed Workload: Critical and Best-Effort Applications

Modern mobile multimedia platforms are characterized by multiple applications, which can be either critical or best-effort. The former are applications, generally known at design time and provided by the device producer/integrator, which implement fundamental tasks for the target device. The latter instead are applications unknown at design time but that each user could add and use once the system is already in production.

The critical applications could be off-line fine tuned to be highly efficient, for example using DSE techniques, and their resource requirement are considered mandatory for a proper functioning of the device. On the contrary, best-effort applications could not be considered at design time, during the definition of the optimization strategy, but their run-time effects could have an impact on the overall system behavior that should be considered by the running optimization policy.

6.5.1.2 Run-Time Phenomena

Let us consider a context of a platform where we admit the presence of multiple applications, running concurrently on the same shared resources, each one having its own *application-specific requirements*. In such a context, the efficiency of managing the available resources is a challenging goal. The mapping of applications onto the available resources may change during the device life-time, and the current effective mapping should be based on specific quantitative metrics, e.g., throughput, memory bandwidth and execution latency. In parallel to the application-specific requirements, we also experience other *non-functional requirements* such as power consumption, energy efficiency and thermal profiles.

The presence of these two types of requirements makes the mapping decision even more complex when we consider some characteristic run-time phenomena such as production process-variation, hot-spots generation, resource failures and workload fluctuation.

6.5.2 Integration Requirements and Goals

The idea to integrate effectively the DSE techniques within an Operating System cannot disregard the problems highlighted in the previous section. Moreover, to provide a general control solution which is acceptable by the Linux community, it is required to integrate the DSE control with the existing frameworks.

To satisfy all these goals, by providing a flexible and efficient OS integration of the DSE control techniques, a hierarchical design should be considered. In such a design, the monitoring, management, control and optimization strategy is operated at different granularity levels. Each granularity level collects requirements from higher level, runs a specific optimization policy, and finally identifies a set of constraints delivered to lower levels.

Such a hierarchical approach requires the development of a new framework, in between the DSE control policy and the in-kernel existing frameworks, which allows to collect application requirements and match them against resources availability. This matching is performed based on a dynamic optimization policy to be developed at different abstraction levels and according to different optimization goals. Run-time optimization policies derived from DSE are considered as coarse grained configuration points, related to the specific off-line profiled application. Nevertheless, such information could be aggregated at run-time with other system requirements in order to achieve a fine-grained and system-wide optimization.

The main *abstraction levels* are user-space, kernel-space and device-space. The user-space level corresponds to the DSE control policy, which can be defined off-line for the fine tuning of profiled critical applications. An example of those kind of policies is presented in [14]. The kernel-level control policy is defined by the new framework, which allows to collect and aggregate requirements from both critical and best-effort applications. This level is somehow similar to the cross-layer frameworks presented in Sect. 6.4.2. Finally, the device-space level is related to each pre-existing subsystem specific control policy like those presented in Sect. 6.4.1.

Regarding the *optimization goals*, the definition of a new kernel-space framework, which could have a system-wide view on run-time system state and resource availability, allows more easily to develop a control policy which is aware of run-time phenomena. This new framework could provide support for optimization goals which are difficult to define off-line, such as resource usage fairness, application performance, power consumption and thermal management.

The new RTRM kernel-space framework proposed for the integration of the DSE defined control policies allows to effectively target two main goals: dynamic resource partitioning and resource abstraction.

6.5.2.1 Dynamic Resource Partitioning

Each application could be associated to a certain “priority level” related to the different impact that the application (either critical or best-effort) could have on the overall user experience. Thus, the RTRM framework should provide a support to handle both:

critical workloads, that have hard requirements in terms of resource usage along with execution behaviors, and best-effort workloads, for which a penalty either does not impact on perceived behaviors or produce a tolerated QoS degradation.

Both classes of applications should access the available resources through a single run-time resource manager framework. Thus, the role of this framework is to account for available resources, and grant access to these resources to demanding application according to their priority level. To efficiently manage this scenario, the resources could be dynamic partitioned, taking into consideration current QoS requirements and resources availability. This dynamic partitioning will allow to grant resources to critical workloads while dynamically yield these resources to best-effort workloads when (and only while) they are not required by critical ones, thus optimize resource usage and fairness.

6.5.2.2 Resource Abstraction

To suitably tackle the run-time phenomena problem, the RTRM kernel-space framework should handle a decoupled perspective of the resources between the users and the underlying hardware. The user applications will see virtual resources but they will not be aware of which of the physical resources are effectively available. At run-time the RTRM will perform the virtual-to-physical mapping according to the current objective function (low power, high performance, etc.) and run-time phenomena (process variation, temporal and spatial temperature gradients, hardware failures and, above all, workload variation).

6.6 Conclusions

The aim of this chapter is to provide the overall picture of the components and goals of a system-wide resource manager. The need for a Run-Time Resource Manager(RTRM) is dictated by the non-deterministic nature of the complex modern applications. This need, in addition, poses new challenges in finding out a low-overhead solution that satisfies all the functional and non-functional requirements. From a performance and QoS point of view, having multiple applications means having at any point of time different perspectives of the entire system.

Our current research takes full advantages from the DSE-based design flow developed within the MULTICUBE project. The coarse grain analysis of the best operating points to be used at run-time will be improved and integrated in a more comprehensive framework, working at the level of the Operating System, capable to provide a fine-grain tuning of the system configuration considering a system-wide optimization perspective. Our research goal is to develop a hybrid centralized/distributed approach, conveying into a hierarchical strategy for managing power consumption/energy requirements. A hierarchical strategy has the benefits of gathering local control and centralized optimal solution to the problem. In addition, such a hierarchical control

will be based on a hardware/software co-design approach, where the solution is based on hardware and software components.

References

1. Bellasi, P., Fornaciari, W., Siorpaes, D.: A Hierarchical Distributed Control for Power and Performances Optimization of Embedded Systems, *Lecture Notes in Computer Science*, vol. 5974. Heidelberg, Springer Berlin, Berlin, Heidelberg (2010). DOI 10.1007/978-3-642-11950-7.
2. Benini, L., Bogliolo, A., De Micheli, G.: A survey of design techniques for system-level dynamic power management, vol. 8. Kluwer Academic Publishers, Norwell, MA, USA (2002)
3. Brock, B., Rajamani, K.: Dynamic power management for embedded systems. Proceedings on IEEE International SoC Conference pp. 416–419 (2003). DOI 10.1109/SOC.2003.1241556
4. Cumming, P.: The TI OMAP Platform Approach to SoC, chap. 5. Kluwer Academic Publishers (2003)
5. Girdwood, L.: Every Microamp is Sacred - A Dynamic Voltage and Current Control Interface for the Linux Kernel. Tech. rep. (2008)
6. Gross, M.: The PM_QoS framework documentation. Proceedings of Linux Symposium (2008).
7. Hsu, C.H., Kremer, U., Hsiao, M.: Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. In: ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design, pp. 275–278. ACM, New York, NY, USA (2001).
8. International Technology Roadmap for Semiconductors, 2009 Edition. Available at http://www.itrs.net
9. Karger, P.A., Safford, D.R.: I/O for Virtual Machine Monitors: Security and Performance Issues. IEEE Security & Privacy Magazine **6**(5), 16–23 (2008). DOI 10.1109/MSP.2008.119.
10. Keating, M., Flynn, D., Aitken, R., Gibbons, A., Shi, K.: Low Power Methodology Manual: For System-on-Chip Design. Springer Publishing Company, Incorporated (2007)
11. King, R.: The CPUfreq framework documentation (2001). URL http://lxr.linux.no/linux+v2.6.34/Documentation/cpu-freq/
12. Lakshmanan, K., Rajkumar, R.: Distributed Resource Kernels: OS Support for End-To-End Resource Isolation. IEEE (2008). DOI 10.1109/RTAS.2008.37.
13. Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1984)
14. Mariani, G., Avasare, P., Vanmeerbeeck, G., Ykman-Couvreur, C., Palermo, G., Silvano, C., Zaccaria, V.: An industrial design space exploration framework for supporting run-time resource management on multi-core systems. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010 pp. 196–201.
15. Murphy, M., Fenn, M., Goasguen, S.: Virtual Organization Clusters. 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing pp. 401–408 (2009). DOI 10.1109/PDP.2009.23.
16. Nollet, V., Verkest, D., Corporaal, H.: A Safari Through the MPSoC Run-Time Management Jungle. Journal of Signal Processing Systems (2008). DOI 10.1007/s11265-008-0305-4.
17. Nowka, K.J., Carpenter, G.D., Brock, B.C.: The design and application of the PowerPC 405LP energy-efficient system-on-a-chip. IBM J. Res. Dev. **47**(5-6), 631–639 (2003)
18. Pallipadi, V.: cpuidle - Do nothing, efficiently... In: Proceedings of Linux Symposium (2007).
19. Pallipadi, V.: The CPUidle framework (2007).
20. Pallipadi, V., Starikovskiy, A.: The ondemand governor: past, present and future. In: Proceedings of Linux Symposium, vol. 2, pp. 223-238 (2006).

21. Paulin, P.: SoC platforms of the future: challenges and solutions. In: MPSoC Forum 2005 (2005)
22. Saputra, H., Kandemir, M., Vijaykrishnan, N., Irwin, M.J., Hu, J.S., Hsu, C.H., Kremer, U.: Energy-conscious compilation based on voltage scaling. In: LCTES/SCOPES '02: Proceedings of the joint conference on Languages, compilers and tools for embedded systems, pp. 2–11. ACM, New York, NY, USA (2002).
23. Shenoy, P., Hasan, S., Kulkarni, P., Ramamritham, K.: Middleware versus native OS support: architectural considerations for supporting multimedia applications. IEEE Comput. Soc (2002). DOI 10.1109/RTTAS.2002.1137378.
24. Unsal, O.S., Koren, I.: System-level power-aware design techniques in real-time systems. Proceedings of the IEEE **91**(7), 1055–1069 (2003). DOI 10.1109/JPROC.2003.814617
25. Venkatachalam, V., Franz, M.: Power reduction techniques for microprocessor systems. ACM Comput. Surv. **37**(3), 195–237 (2005). DOI http://doi.acm.org/10.1145/1108956.1108957.
26. Yermalayeu, S., Vervoort, G., Mahadevan, S., Becking, B.: Linux clock management framework (2007)
27. Zhao, S., Chen, K., Zheng, W.: The Application of Virtual Machines on System Security. IEEE (2009). DOI 10.1109/ChinaGrid.2009.45.

Part II
Application Domains

Chapter 7
High-Level Modeling and Exploration of a Powerline Communication Network Based on System-on-Chip

Marcos Martinez, David Ferruz, Hector Posadas, and Eugenio Villar

Abstract This chapter will present the application of MULTICUBE methodology to the design of a ITU G.hn compatible component for powerline communication. Powerline communication is an advanced telecommunication system allowing fast and reliable transfer of audio, video and data information using the most ubiquitous transmission system: the power lines. This transmission line is used to exchange information between different equipments connected to the network by using advanced coding techniques like such as Orthogonal Frequency Division Multiplexing. The starting point of the analysis is a high level SystemC-based virtual platform where we will study the effects of the variation of a pre-defined set of design parameters on a set of pre-defined metrics. This automatic analysis will drive the design choices in order to build an optimized industrial system. We will show that the SystemC-based virtual platform combined with the MULTICUBE design space exploration framework can save up to 80% of designer work time while achieving better results in terms of performance.

7.1 Introduction

In this chapter we will focus on the application of MULTICUBE methodology to a DS2 use case in order to stress the proposed design exploration flow in a real industrial environment. This use case is based on the high-level modeling and analysis of an in-home transmission system over powerline infrastructure based on the novel ITU-G.hn standard. The model will be analyzed in order to optimize Quality of Service and design parameters. The resulting automatic analysis flow will be compared to current semi-automatic approaches based on scripting techniques.

The chosen use case shows the integration of MULTICUBE flow with a high level modeling platform written in SystemC with TLM-based communications where

H. Posadas (✉)
University of Cantabria, Santander, Spain
e-mail: posadash@teisa.unican.es

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_7, © Springer Science+Business Media, LLC 2011

several parameters can be modified in order to optimize the performance of the overall system and different design metrics.

The physical links between the different nodes of a powerline network present different characteristics depending on factors like distance between the nodes, electrical appliances connected to the network, quality of the infrastructure, etc. This is translated to the fact that the transmission speed, latency and QoS can depend heavily on the route selected to transmit information between two nodes.

Many of the design parameters of the system have to be dimensioned taking into account the application, the physical effects, noise, etc. This requires many simulations to be run in order to find the best value for each of these parameters. However, these simulations require a long real time in order to make appear some of the effects we want to investigate. This way, it is not possible to work at RTL level and thus, the use of high level models is mandatory.

This virtual platform approach has been used in past company's designs in a semi-automatic optimization flow but, in this study, we will improve the system by applying MULTICUBE fully automatic techniques. This way, we will be able to compare the results that are obtained in both approaches in terms of performance (how good the chosen metrics are) and process optimization (how fast we can converge towards an optimum solution) while minimizing the human intervention in the process.

For that, the use case provided by DS2 will be run through MULTICUBE flow described in Chap. 1. Chap. 2 introduced the use of MULTICUBE-SCoPE plug-in from the University of Cantabria for modeling and integration of the platform. This plug-in will be used in DS2 use case along with modeFRONTIER DSE tool (from ESTECO) in order to optimize the above-mentioned parameters.

7.2 Design Case Study: High Level Modeling of a Powerline Network Based on SoC

7.2.1 Application Description

7.2.1.1 Configuration of the System Components

Design of Systems on Silicon (DS2) is a fabless silicon design house and one of the leading suppliers of silicon and software for Power line Communications (PLC). Founded in 1998, DS2 produced first time working silicon and is currently marketing chipsets with performance rates of up to 200 Mbps. DS2 technology converts the existing electricity grid and domestic power lines into a high-speed communication network supporting voice, video and data services, at low cost. DS2 technology not only delivers high performance but it is also flexible and efficient supporting co-existence between the PLC access and LAN markets. In the framework of the design space exploration project MULTICUBE, as an industrial test case, DS2 has used a complete model of its next generation Powerline communication system (DS2 assets and technologies were recently bought by Marvell Technology Group).

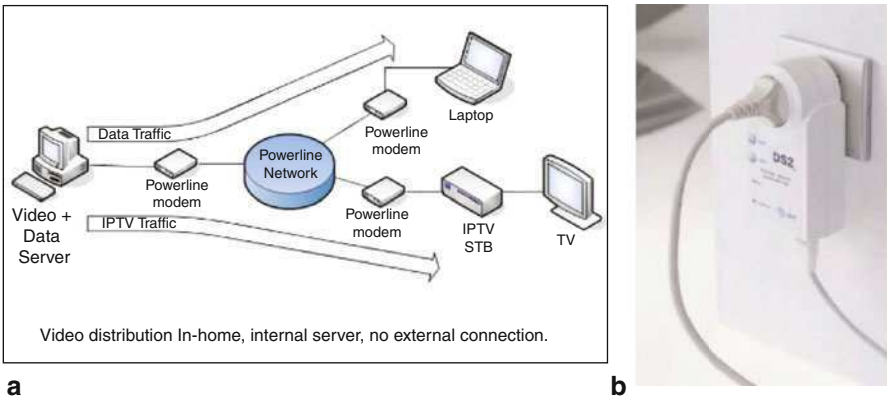

Fig. 7.1 Example of a typical PLC application (**a**) and PLC modem (**b**)

PLC stands for "*Powerline Communication*", an advanced telecommunications system allowing fast and reliable transfer of audio, video and data information using the most ubiquitous transmission system: the power lines (Fig. 7.1). This transmission line is used to exchange information between different equipment connected to the network using the most advanced coding techniques like OFDM ("*Orthogonal Frequency Division Multiplexing*").

PLC technology has different flavors and implementations, depending of the application domain where it is used. For each of these domains, adapted PLC equipment needs to be designed and deployed. The three most important applications domains nowadays are:

- **Automatic Meter Reading (AMR) / Automatic Metering Infrastructure (AMI):** These systems are used by the electrical utilities to monitor the real-time evolution of the electrical demand in order to manage the electrical distribution network in an efficient way. As the number of data to be transmitted is low, usually narrow-band PLC systems are used in this context.
- **Broadband Access PLC:** Use of the electrical grid to connect the Internet backbone to the end user in order to provide broadband information through the electrical wiring. This technology is mainly used by electrical utilities to provide cheap Internet access to their customers and for core applications for their own telecommunications needs. Finally, this technology is also being deployed in order to extend other access technologies (mainly fibers) inside buildings.
- **In-Home PLC:** Use of the domestic electrical grid to connect different audio/video/data home equipment with a high speed, high quality link (up to 200 Mbps at the physical layer). The main use of this technology is the high quality Audio/Video distribution within the home for IPTV extensions (received by any access technology (xDSL, cable, xPON, PLC, . . .) and home media servers.

In early 2010, a new technology has been standardized under ITU umbrella. This new standard ("*ITU G.hn*") allows the use of different physical media (powerline,

coaxial, phone line) over which we can achieve multiple parallel high definition video transmissions through the houses. For each of the physical layers, an optimized set of parameters will be used, demonstrating the necessity of a powerful optimization tool for fine tuning the chipset for each application. The powerline use case described in this chapter is based on a high level description of such new G.hn standard.

In MULTICUBE project, DS2 makes use of a virtual platform that describes at a high level of abstraction both the hardware and software components of a G.hn system. This platform, named *STORM*, models such a new system in high level SystemC-TLM and making use of SCoPE technology for HW/SW integration. In this model, all the parameters can be configured in order to stress them and find the most suitable combination.

7.2.1.2 Description of the Use Case

The high level model of an "*ITU G.hn*" system used as test case implements a network of several powerline nodes connected through a powerline channel. These PLC nodes include the different layers that are described in the standard, focusing specially on the MAC layer, where the main design decisions have to be taken.

The model is adapted to the future hardware platform that will be used in the real platform-based ASIC implementation where an embedded microprocessor will run a PLC SW stack code that will make use of different hardware resources, accessible through a bus system. A simplified (with only four nodes) high level view of such a platform is shown in Fig. 7.2 and can be described as follows:

- **Ethernet traffic generators & monitors:** These modules generate UDP/TCP flows that will be injected through the Ethernet port of the corresponding PLC

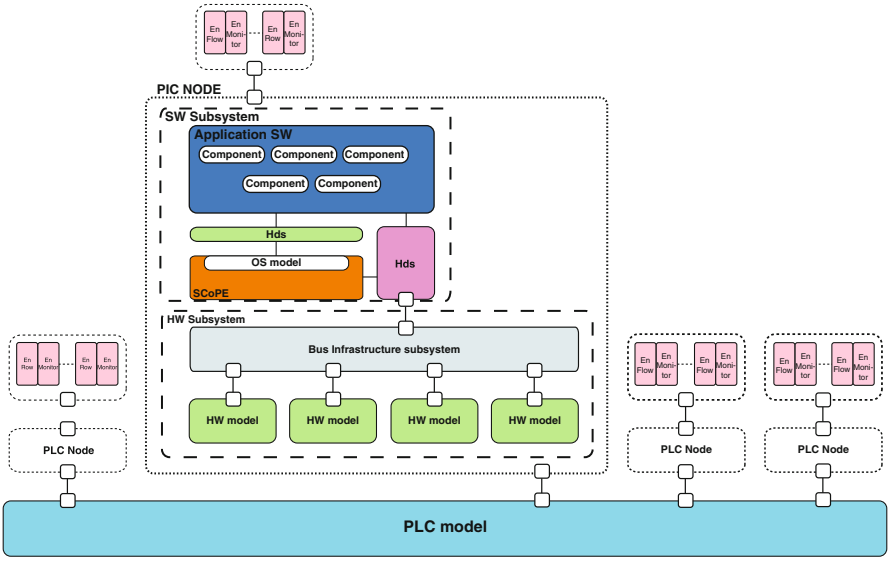

Fig. 7.2 PLC system high level description

node in order to model real traffic. Each generator is coupled with a traffic monitor. The number and characteristics (UDP/TCP, traffic shape, etc.) can be configured in the simulation.

- **PLC channel:** This module models the behavior of a real PLC channel, introducing link characteristics (noise, etc.) to each of the logic connections between the different nodes of the system. The characteristics of the PLC channel can be configured in the simulation.
- **PLC Nodes:** This module represents the part of the system that will be built over a DS2 chipset. It is divided in the following components:
  - *HW models:* High level model of hardware blocks in the final system. Each of the modules is written in C and presents SystemC-TLM ports to be connected to the bus and the rest of the modules. The model respects the timings of the final implementation and has been annotated with important information for the simulation (power consumption, resources, etc.)
  - *Bus interconnection infrastructure:* SystemC/TLM model of the bus that will be present within DS2's digital circuit
  - *HdS:* Hardware-dependent software is part of the overall software. This block represents the lower levels of the software, making the interface with hardware resources. As we can work at different levels of abstraction for the HW models (TLM to RTL). The HdS has to be updated for each of the levels of abstraction we are working with (register-level when working with RTL and transaction level when working in SystemC). In the case of MULTICUBE, a simplified version of the HdS is used since we only work at system level
  - *OS Model / SCoPE:* Through the use of SCoPE tool, we can model our real time Operating System (OS) in SystemC-TLM and make use of it in our platform. This way, we can run application software directly in the platform without having to adapt it and without making use of a slow ISS for the microprocessor behavior. SCoPE also introduces an estimation of the timings of the software and synchronizes SW timings and HW timings. MULTICUBE SCoPE plug-in extends SCoPE functionalities that we will see later on.
  - *Application SW:* Real application SW that will be run over the virtual platform, the FPGA debugging prototypes and final ASIC products. In MULTICUBE, the SW will be frozen

While other use cases of the project are focused on applications that look for the highest computing power with the available resources, powerline use case is centered in optimizing a medium-end platform that run a control-oriented complex protocol. The overall goal is to optimize the cost of the overall solution while maintaining the necessary performance for the specific application but with minimal changes in the platform processing power. The main differences with the other use cases presented in MULTICUBE project are listed in Table 7.1.

Table 7.1 Particularities of DS2 use case

Particularity	Effect on the analysis
The platform is less computing-intensive than the other use cases defined within the project	The metrics to analyze are different from the other use case
The focus of the optimization phase is put on cost-effectiveness of the final solution and not really on its final performance	The metrics are more difficult to measure since they can rely on subjective analysis
The structure of the use case is data flow oriented. The optimization is done over the control of this data flow	The simulation results rely strongly on the data flow contents and behavior
Many parameters of the platform are fixed by the standard or by legacy hardware	Less freedom to choose some of the parameters
The optimization procedure was already semi-automatic in past designs	The evaluation procedure is different from other use cases

7.2.2 Platform Description

7.2.2.1 STORM Platform Description

One of the applications that will be used for testing the new developments in MULTICUBE will be a communication system based on the novel ITU-T G.hn wired networking standard ratified early 2010. This effort is aimed to provide support for networking over power lines, phone lines and coaxial cables using the same standard and similar components. For each of the physical layers, an optimized set of parameters will be used, demonstrating the necessity of a powerful optimization tool for fine tuning the chipset for each application. In MULTICUBE project, DS2 makes use of a virtual platform that describes at a high level of abstraction both the hardware and software components of a G.hn system. This platform, named STORM, models such a new system in high level SystemC-TLM making use of SCoPE technology for HW/SW integration. In this model, all the parameters can be configured in order to stress them and find the most suitable combination.

The platform created is a fundamental step in the process of creating a new chipset for powerline communication following ITU G.hn standard. In this sense, the goal of this platform is multiple, considering the needs of all the persons that are using it. These needs are listed in Table 7.2.

Each of these actors are looking for a different result from the platform optimization, however, each of them can make use of the methodologies advancements provided by MULTICUBE. In any case, for the purposes of this demonstration, we only focus on the system designer's perspective since they will benefit the most with the metric optimization procedures provided by the project.

As mentioned before, G.hn layers are specified in the STORM platform with a SystemC-TLM. The following rules have been followed in order to maximize the

Table 7.2 Actors participating to virtual platform analysis

Actor	Use	Goal
SW designers	Develop application Software over the virtual platform instead of waiting for the FPGA prototype or the ASIC design	Speed up design process
HW designers	Fine tune the hardware sub-system and the interactions between the hardware and the software. Proceed with the refinement process	Optimize HW blocks and the interface between HW and SW
System designers	Explore architectural approaches and freeze design parameters	Obtain the optimum design and architecture

reuse of the different components, increase the flexibility of the platform and to speed-up the learning-curve of the model.

- All the communication interfaces of the components of the model are described in high level SystemC using TLM libraries
- All the behavior of the components of the model are described in plain C code in order to facilitate HW/SW partitioning
- TLM libraries have been tuned by DS2 in order to monitor transactions, facilitate the reuse of modules and reduce the learning curve of the model
- The platform has been designed to be highly configurable and in order to reuse as many blocks as possible

In order to stress as many parameters as possible and to cope with all possible situations, the current version of the platform is highly configurable by itself through the use of proprietary ASCII configuration files that can modify the behavior of the main components of the system, independently of their hardware or software nature. In MULTICUBE flow, some of these parameters are overwritten directly by MULTICUBE configuration files in order to automatically control the optimization process. In the use case, and in order to make comparisons possible, we have modified the platform to run either in stand-alone mode (using only ASCII configuration files as input) or in MULTICUBE mode using the project configuration files in addition to the ASCII configuration files.

STORM platform has been designed with the objective to be as general, flexible and open as possible and do not rely on commercial tools for running. In this sense, SystemC and TLM were chosen as the backbone of the platform but, thanks to the flexibility of the platform, other languages (VHDL, Verilog, SystemVerilog, C, C++, SCV, etc.) can be added when needed to investigate particular aspects of the flow or different abstraction levels.

For the purposes of MULTICUBE project, we will focus on the system level for optimizing the system parameters of the platform. This way, only SystemC-TLM components have been considered and no RTL code has been included. However, it is interesting to mention that the whole process can be also run over other

implementations of the platform including low-level refined components written in Verilog.

Thus, in addition to MULTICUBE tools that will be mentioned later in this study, the relevant tools used in the platform and that have an impact on this study are the following:

- **SCoPE:** SCoPE tool allows to model Real Time Operating Systems (RTOS) into TLM and to describe the interface between hardware and software. Within STORM, SCoPE is aimed to allow application SW and MicroC-OS II operating system over the virtual platform
- **DS2_SC:** This library is the basis of the platform and contains both introspective features and communication features. MULTICUBE interface (SCoPE plug-in) makes use of the API provided by the library in order to inject the parameters defined by MULTICUBE configuration file in the platform

7.2.2.2 Application of MULTICUBE Tools in the Platform

Figure 7.3 describes the interactions between the different elements that compose the flow.

The three main entities that interact in DS2 use case are:

- **STORM platform:** Model of an ITU G.hn MAC level already described in previous sections
- **MULTICUBE SCoPE:** Extension of SCoPE tool that has a double objective: Describe the platform from an XML entry point and implement the interface between the platform and MULTICUBE exploration tools
- **modeFRONTIER:** design exploration and analysis tool that controls the whole process

In the following sections we will comment in detail on the role and particularities of these tools.

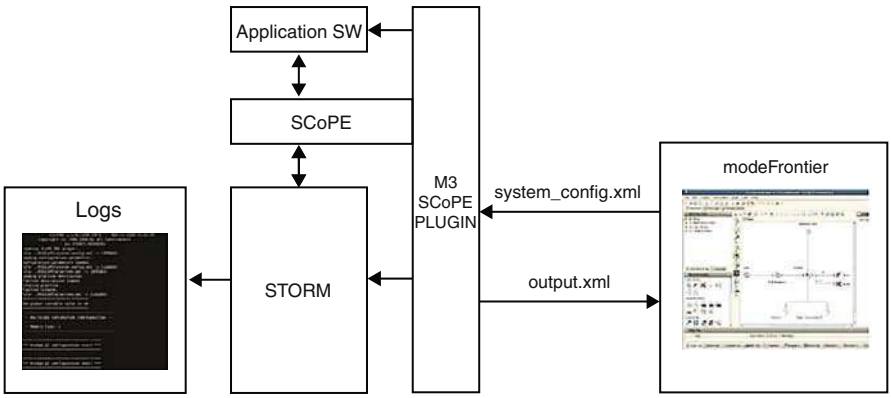

Fig. 7.3 Interactions between the different use case actors

7.2.2.3 modeFRONTIER: Analysis and Exploration Tool

modeFRONTIER tool from ESTECO acts as the control entity of the process. Through the MULTICUBE-defined interface, it injects chosen combinations of parameters in the platform. MULTICUBE SCoPE plug-in receives these combinations and modifies the platform behavior. The results are transmitted back to modeFRONTIER through another MULTICUBE-defined interface.

In order to define the whole process, modeFRONTIER needs to be informed about the parameters and metrics that are being studied through a MULTICUBE-defined XML file. In this use case, we have used the following file:

```
<?xml version="1.0" encoding="UTF-8"?>
<design_space xmlns="http://www.multicube.eu/" version="1.3">
   <simulator>
     <simulator_executable path="/local/projects/External/
     Multi3/UseCase/v6_0/test/system/test1/run.sh" />
   </simulator>
   <parameters>
     <parameter name="LOSS_value" description="% loss of
     packets" type="integer" min="0" max="10000" step="50" />
     <parameter name="MAX_CELLS_PKT_value" description="MAX
     CELL PKT" type="integer" min="64" max="4096" step="32"/>
     <parameter name="MAX_CELLS_FEC_value" description="MAX
     CELL FEC" type="integer" min="64" max="4096" step="32"/>
     <parameter name="FEC_LENGTH_value" description="FEC
     LENGTH" type="integer" min="120" max="540" step="420"/>
     <parameter name="ACK_value" description="ACK ACTIVE"
     type="integer" min="0" max="1" step="1" />
   </parameters>
   <rules>
   </rules>
   <system_metrics>
     <system_metric name="Power_Consumption" type="float"
     unit="W"/>
     <system_metric name="xput0_1" type="float"
     unit="Bit/s"/>
     <system_metric name="xput1_0" type="float"
     unit="Bit/s"/>
     <system_metric name="MaxJitter0_1" type="float"
     unit="s" desired="small"/>
     <system_metric name="LatencyMin0_1" type="float"
     unit="s" desired="small"/>
     <system_metric name="LatencyMax0_1" type="float"
     unit="s" desired="small"/>
     <system_metric name="LatencyMean0_1" type="float"
     unit="s" desired="small"/>
     <system_metric name="MaxJitter1_0" type="float"
     unit="s" desired="small"/>
     <system_metric name="LatencyMax1_0" type="float"
     unit="s" desired="small"/>
     <system_metric name="LatencyMin1_0" type="float"
     unit="s" desired="small"/>
     <system_metric name="LatencyMean1_0" type="float"
```

```
        unit="s" desired="small"/>
      </system_metrics>
    </design_space>
```

All these parameters and metrics will be further explained in following sections of this study.

As we can see in the dse.xml file, we explicitly specify to modeFRONTIER which is the command that has to be launched by the tool in order to run a single simulation. If we take a closer look to this file, we have:

```
#!
echo "---> Running simulation"
$EXAMPLE_PATH/storm -xml $EXAMPLE_PATH/platform.xml $1 \
-xmd $EXAMPLE_PATH/metrics.xml $2 \
mv output.xml system_metrics.xml
```

In this bash shell we can identify three types of information passed to the platform:

- The configuration files from the platform itself (which are implicit in the storm executable)
- The configuration files that drive the behavior of M3SCoPE (platform.xml and metrics.xml) and that are fixed for all the simulations
- The configuration files that are fixed by modeFRONTIER ($1 and $2). The first parameter is the system configuration file (in this case system_config.xml) and the name of the output file to be used by the platform (in this case system_metrics.xml)

The last line of the bash file makes only some adaptation in the output XML file to be read by modeFRONTIER.

7.2.2.4 M3-SCoPE: Interface and Description Tool

Once the executable is run, we have still to inject the parameters extracted from the interface file into the internals of the platform. This is done through the use of MULTICUBE-SCoPE tool developed within the project. This tool has been built as a plug-in of the above-mentioned tool SCoPE and its modeling capabilities are described in Chap. 2 of the book.

M3-SCoPE needs different configuration files to run:

- **platform.xml:** This file describes in a proprietary format the parameters that have to be read from the interface file created by modeFRONTIER. It may be used also to describe the topology of the top level. However, for the purpose of this test, the architectural description is already hard coded in the SystemC virtual platform
- **metric.xml:** This file describes the output metrics that M3-SCoPE will extract at the end of each simulation and is included in the interface file that will be sent towards modeFRONTIER. It is interesting to note that for the purposes of this use case, the output metrics are either generated by M3-SCoPE or directly created by the platform itself (using DS2_SC libraries). In the end, both results are concatenated and passed to modeFRONTIER

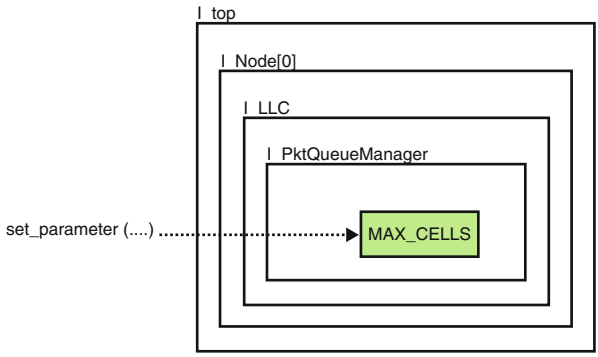

Fig. 7.4 Injection of parameters in the virtual platform

Table 7.3 Injection of parameters in the platform

cout < < "MULTI3 : Creating SCoPE XML plugin..." < < endl; int status = uc_xml_init_plugin(argc, argv);	Get the parameters value using MULTICUBE-SCoPE
I_top->I_Node[0]->I_LLC->i_PKTQueueManager->set_parameter("MAX_CELLS", ss_MAX_CELLS_PKT.str());	Use DS2_SC's API to inject parameter in the platform

Through MULTICUBE-SCoPE the platform first obtains its parameters to be read. Then, the plug-in reads the configuration file from modeFRONTIER and associates a value to the parameter.

Next step is then to use DS2_SC APIs in order to modify the value of the internals of the platform as shown in Fig. 7.4.

To do so, the lines of code displayed in Table 7.3 are used.

Once a simulation finishes, the platform generates an XML file that is given as a feedback to the modeFRONTIER tool. An example of such a file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<simulator_output_interface xmlns="http://www.multicube.eu/"
 version="1.3">
     <system_metric name="Power_Consumption"
     value="1.752689280e-01"/>
     <system_metric name="xput0_1" value="4.05376e+07"/>
     <system_metric name="LatencyMin0_1" value="822.944"/>
     <system_metric name="LatencyMax0_1" value="7107.76"/>
     <system_metric name="LatencyMean0_1" value="22104.8"/>
     <system_metric name="MaxJitter0_1" value="6284.82"/>
     <system_metric name="xput1_0" value="810158"/>
     <system_metric name="LatencyMin1_0" value="4114.55"/>
     <system_metric name="LatencyMax1_0" value="11745.5"/>
     <system_metric name="LatencyMean1_0" value="17720"/>
     <system_metric name="MaxJitter1_0" value="7630.98"/>
</simulator_output_interface>
```

Finally, the last step of the process is to analyze the results thanks to the analysis facilities provided by modeFRONTIER. In later sections of this study we will show the main results of the process.

7.2.2.5 Description of Parameters and Metrics

The use case platform has been configured in order to continuously transmit IP flows between two of the nodes of the system (which is composed of up to 16 PLC nodes). The configuration of this flow is the following:

- **Simulation characteristics**
  - Simulation length is 0.5 s of real time
  - The parameters that are not studied are kept constant and at a "normal" value
- **Topology of the network**
  - Sixteen Nodes are active during the whole simulation (all the nodes are running and listening)
  - The communication is done only between two of the nodes of the system: Node 0 and Node 1
  - Channel noise can be modified by MULTICUBE tools to be between 0 and 100% probability of loss
- **Traffic shape**
  - Traffic type is multimedia (MPEG) generated by the platform traffic generators
  - Traffic is monitored and the quality of the link is continuously checked
  - The communication is bidirectional
  - All the flows are UDP-based
  - Flow rates are kept constant. No peaks in the flow have been configured
  - The IP flow we inject in the network from Node 0 to Node 1 is 40 Mbps
  - The IP flow we inject in the network from Node 1 to Node 0 is 1 Mbps
- **Quality of Service**
  - No prioritization of traffic between nodes
  - No prioritization of flows within a single node

With this setup, we have defined a set of parameters that have been analyzed during this use case and are presented in Table 7.4.

It is interesting to note that four out of these five parameters are design parameters while the fifth one (LOSS) is an ambient parameter. This means that while the first four parameters have to be optimized, the fifth one is there to put the use case under different situations for the analysis.

All these parameters are related to the main concepts we want to investigate in the design: Memory length and QoS capabilities. None of these concepts is more important than the other and these concepts will depend on the final application of the integrated circuit (mainly because the optimization process was long and thus, there was only time to optimize one of them). Till now, the procedure was to guarantee a correct performance for all concepts. Thanks to MULTICUBE processes and the automation of the procedure we can envision to design more adapted ICs for each situation.

In addition to the parameters, a set of metrics have been used to evaluate the performance of the system under different running conditions. These metrics are presented in Table 7.5.

Table 7.4 Design parameters analyzed in the use case

Parameter	Value	Unit	Related concept	Description
MAX_CELL (PKT)	64–4096	Cells	Memory (queue length)	Maximum number of packets that are stored in the modem internal memory
MAX_CELL (FEC)	64–4096	Cells	Memory (queue length)	Maximum number of FEC values that are stored in the modem internal memory
ACK ON/OFF	ON/OFF	Boolean	QoS Policy	Packet Acknowledgement protocol activated or not
PHY_BLOCK_SIZE	120/540	Bytes	Memory	This parameter is called FEC_LENGTH_value in the dse.xml and describes the length of the FEC block in the physical packet sent through powerline channel
LOSS	0–100	%	Channel characteristics	Percentage of lost packets while sent through the physical medium

Table 7.5 Design metrics analyzed in the use case

Parameter	Value	Related concept	Description
xPut (Node 0 → Node 1)	Bits/s	Flow bandwidth	Measured throughput between Node 0 and Node 1
xPut (Node 1 → Node 0)	Bits/s	Flow bandwidth	Measured throughput between Node 1 and Node 0
Max latency (Node 0 → Node 1)	ms	QoS	Maximum latency measured in the Node 0 to Node 1 transmission
Min latency (Node 0 → Node 1)	ms	QoS	Minimum latency measured in the Node 0 to Node 1 transmission
Mean latency (Node 0 → Node 1)	ms	QoS	Average latency measured in the Node 0 to Node 1 transmission
Max latency (Node 1 → Node 0)	ms	QoS	Maximum latency measured in the Node 1 to Node 0 transmission
Min latency (Node 1 → Node 0)	ms	QoS	Minimum latency measured in the Node 1 to Node 0 transmission
Mean latency (Node 1 → Node)	ms	QoS	Average latency measured in the Node 1 to Node 0 transmission

The parameters are analyzed for the two directions of the traffic: from Node 0 to Node 1 and from Node 1 to Node 0 since no prioritization is done. Thanks to MULTICUBE flow, we will try to minimize latencies while maintaining the throughput and with as less resources as possible.

7.2.3 DSE Assessment Process

The general idea is to make use of the configuration files provided in STORM platform and that parametrize almost all the possible parameters of the system. As shown in Fig. 7.5, a set of parameters are selected to be optimized from the whole number of possible parameters. In the same sense, a set of metrics is selected for the analysis. These parameters and metrics are the ones presented in precedent sections of this study.

The study presented in this chapter compares two optimization procedures that can be used in order to freeze the design parameters: the traditional semi-automatic procedure normally used in DS2 and the MULTICUBE optimization flow proposed in the project. Both approaches will be presented and commented in the following sections and are based on the common optimization flow that was defined by all the partners in MULTICUBE (Fig. 7.6) as described in Chap. 1.

With this starting point, the optimization procedure has been adapted to DS2 specific context, obtaining the procedure shown in Fig. 7.7.

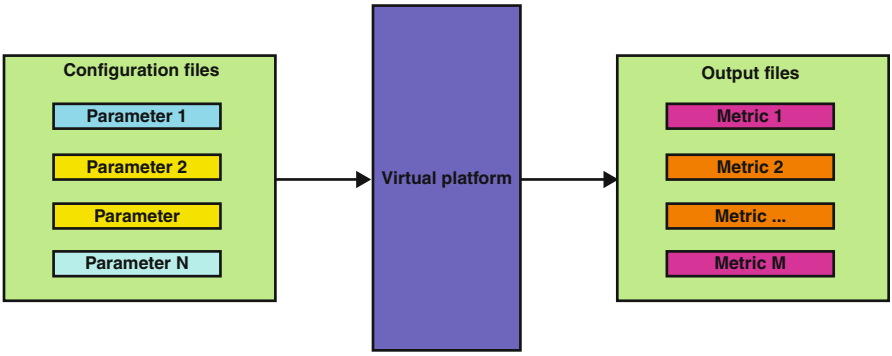

Fig. 7.5 Relationship between parameters and metrics

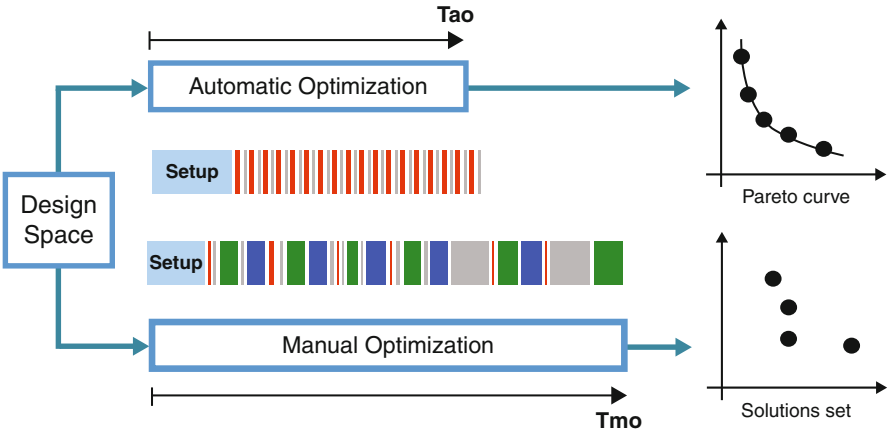

Fig. 7.6 Comparison between manual and automatic optimization

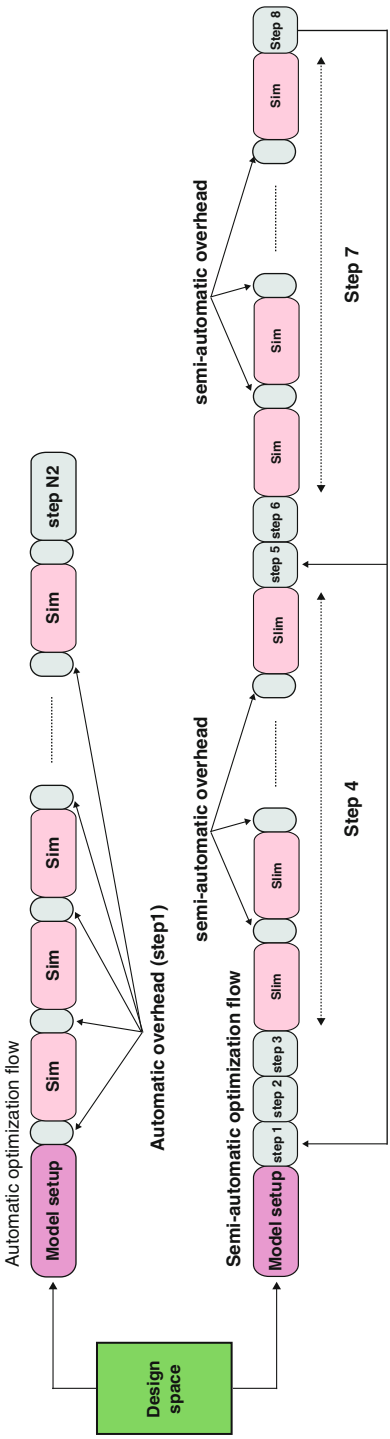

Fig. 7.7 Optimization procedures followed in the project by DS2

Table 7.6 Semi-automated procedure steps

STEP	Description
Step 0	Build a configurable platform using ASCII files (DS2 proprietary lan-guage) to configure the platform. Prepare the platform to overwrite some of these configurations by MULTICUBE own tools
Step 1'	Automated parameter update procedure and metric extraction
Step 2'	Final analysis of the solutions in order to select the optimized combination of parameters

- **Semi-automated procedure (or "traditional procedure")**. In order to optimize different chipsets, DS2 has developed an in-house semi-automated procedure. This procedure is possible thanks to the high flexibility and configurability of the STORM platform. Using this platform, the optimization procedure is performed by following the steps shown in Table 7.6.
- **Automated procedure (MULTICUBE flow)**. MULTICUBE procedure has been applied to DS2 use case in order to optimize the above mentioned parameters. As a consequence, the number of steps have been reduced significantly. The resulting flow is presented in Table 7.7. The procedure has been run on a dual-core desktop PC with Ubuntu 8.04 OS for 36 h under the supervision of a team of two engineers. Thanks to MULTICUBE automation, up to 534 experiments have been done without human intervention. In order to achieve this result we have setup the experiments using the elements of the modeFRONTIER workflow (Fig. 7.8). In this workflow, we have selected the points to analyze in two ways: one by choosing an initial Design of Experiments (DoE) composed of a random sampling and another by using MOGA-II algorithm (Multi Object Generic Algorithm) as

Table 7.7 Automated procedures steps

STEP	Description
Step 0	Build a configurable platform using ASCII files (DS2 proprietary lan-guage) to configure the platform
Step 1	Select a subset of 2–3 parameters and 2–3 metrics
Step 2	For the non-selected metrics, fix the rules of what are the accepted boundaries
Step 3	Write automation scripts to generate all selected combinations and to filter the requested metrics from the output files and facilitate manual in-spection
Step 4	Run simulations for all possible selected combinations with the automated script and automatically extract metrics from the simulation results
Step 5	Manual inspection of metrics. Selection of a set of the parameter com-bination that seems to be the better adapted. Selection is done through designer's experience
Step 6	Generate new platforms with the selected combination of parameters and corner cases for the non-selected parameters
Step 7	Run simulations with the new set of platforms
Step 8	Verify that the ALL the metrics meet the rules specified in step 2. If not, Investigate causes. If the cause can be corrected by changing one of the selected parameters select a new combination in step 5. If the cause cannot be corrected by changing one of the selected parameters Change the selected combination of parameters (include new ones) and come back to step 1

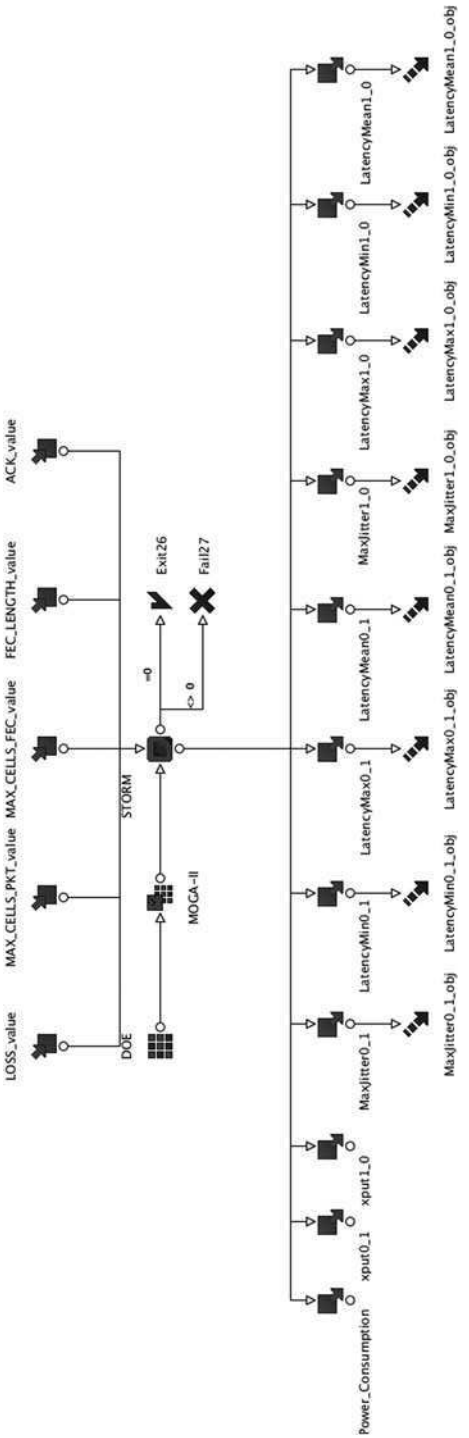

Fig. 7.8 modeFRONTIER workflow

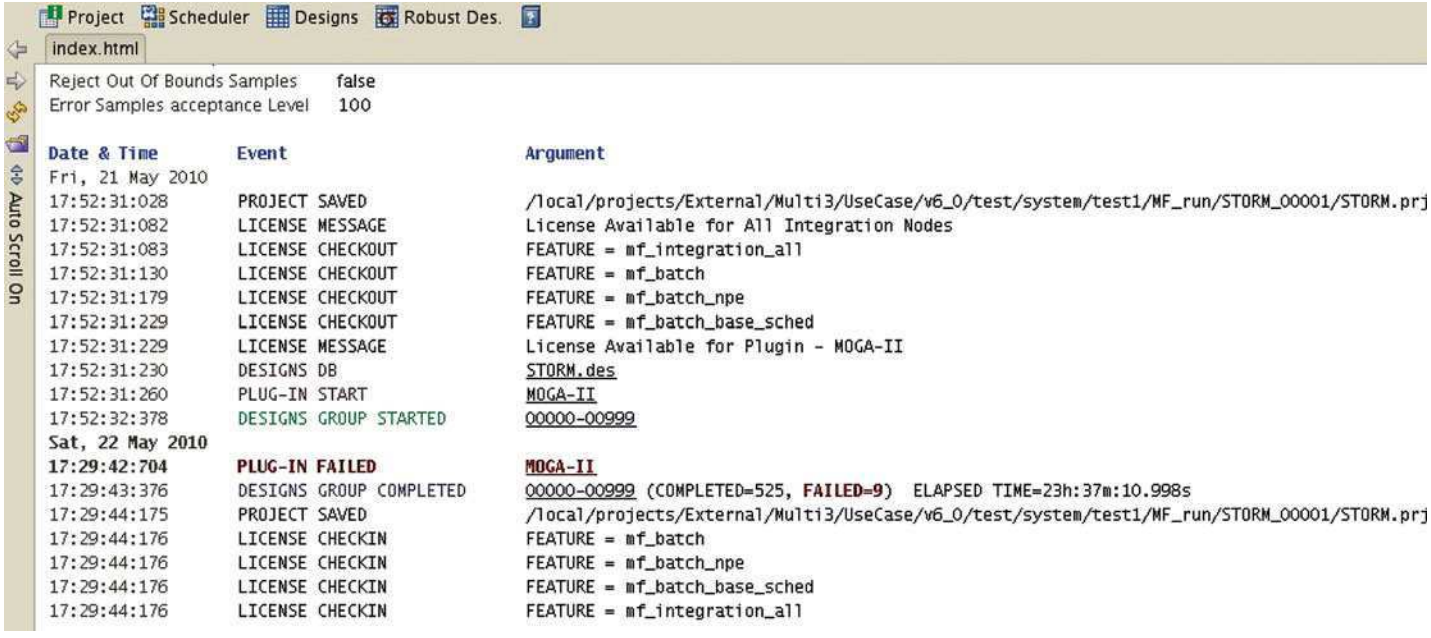
Project
Scheduler
Designs
Robust Des.
index.html
Auto Scroll On
Reject Out Of Bounds Samples false
Error Samples acceptance Level 100
Date & Time Event Argument
Fri, 21 May 2010
17:52:31:028 PROJECT SAVED /local/projects/External/Multi3/UseCase/v6_0/test/system/test1/MF_run/STORM_00001/STORM.prj
17:52:31:082 LICENSE MESSAGE License Available for All Integration Nodes
17:52:31:083 LICENSE CHECKOUT FEATURE = mf_integration_all
17:52:31:130 LICENSE CHECKOUT FEATURE = mf_batch
17:52:31:179 LICENSE CHECKOUT FEATURE = mf_batch_npe
17:52:31:229 LICENSE CHECKOUT FEATURE = mf_batch_base_sched
17:52:31:229 LICENSE MESSAGE License Available for Plugin - MOGA-II
17:52:31:230 DESIGNS DB STORM.des
17:52:31:260 PLUG-IN START MOGA-II
17:52:32:378 DESIGNS GROUP STARTED 00000-00999
Sat, 22 May 2010
17:29:42:704 PLUG-IN FAILED MOGA-II
17:29:43:376 DESIGNS GROUP COMPLETED 00000-00999 (COMPLETED=525, FAILED=9) ELAPSED TIME=23h:37m:10.998s
17:29:44:175 PROJECT SAVED /local/projects/External/Multi3/UseCase/v6_0/test/system/test1/MF_run/STORM_00001/STORM.prj
17:29:44:176 LICENSE CHECKIN FEATURE = mf_batch
17:29:44:176 LICENSE CHECKIN FEATURE = mf_batch_npe
17:29:44:176 LICENSE CHECKIN FEATURE = mf_batch_base_sched
17:29:44:176 LICENSE CHECKIN FEATURE = mf_integration_all

Fig. 7.9 Powerline use case imported in modeFRONTIER

optimization strategy. Once the optimization flow is run, we can see the results of the different experiments implemented (Fig. 7.9). The first result is that out of all 534 designs, nine designs have failed because they drove the platform to corner cases that made the virtual platform to an unstable state. This unforeseen result is quite interesting since it demonstrates that the flow will also flag any possible issues in the platform itself. Moreover, regarding the 525 completed experiments, it is interesting to note that the analysis has helped the design team in two different processes: to demonstrate the coherency of the results and to find the best point for the design parameters.

7.2.3.1 Demonstrate the Coherency of the Results

The first thing to be analyzed with the procedure followed in MULTICUBE is the coherency of the obtained results taking into account the real industrial application and the expected theoretical results. The best way to proceed in the case of a powerline multimedia transmission is to consider the diagrams of the loss in the path versus the throughput and latency in each of the directions of the communication as shown in Figs. 7.10–7.13.

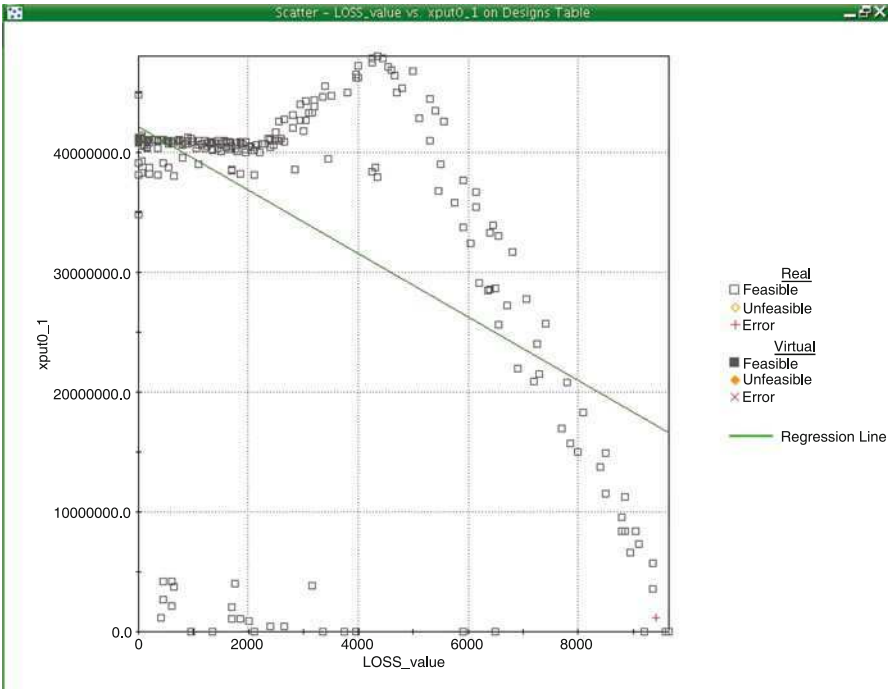

Fig. 7.10 Loss vs throughput, from Node 0 to Node 1

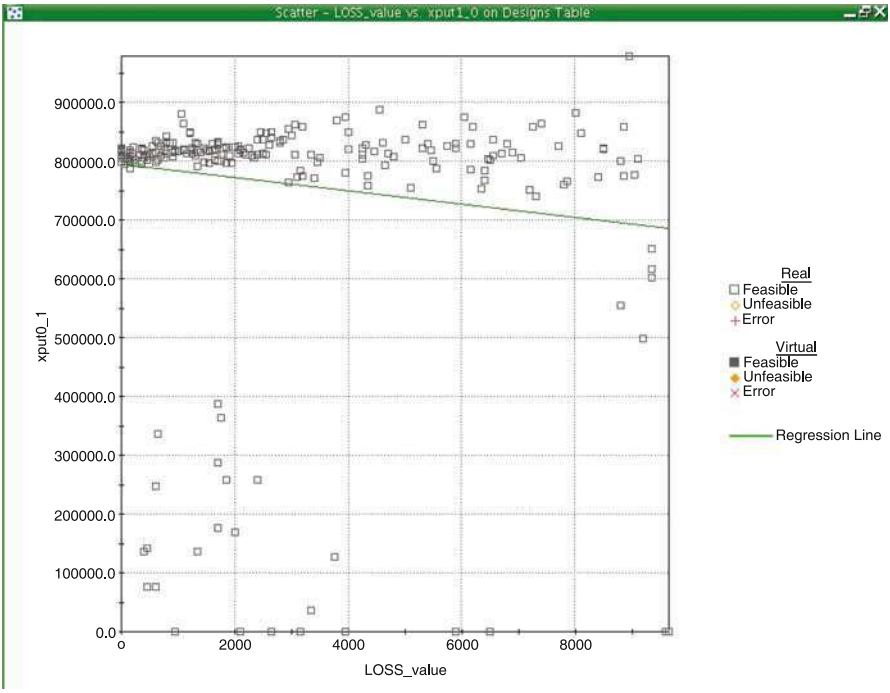

Fig. 7.11 Loss vs throughput, from Node 1 to Node 0

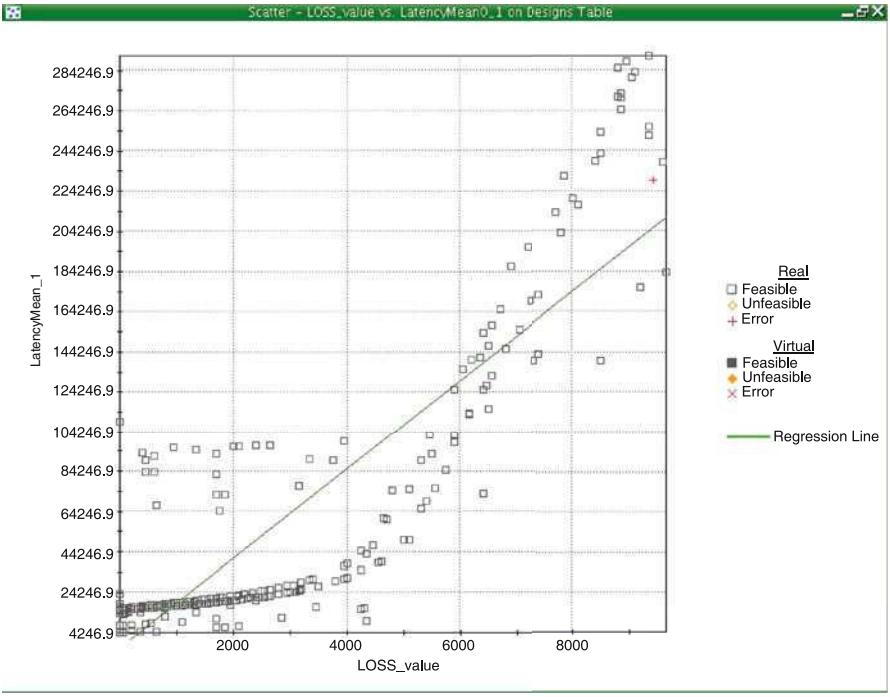

Fig. 7.12 Loss vs Mean Latency, from Node 0 to Node 1

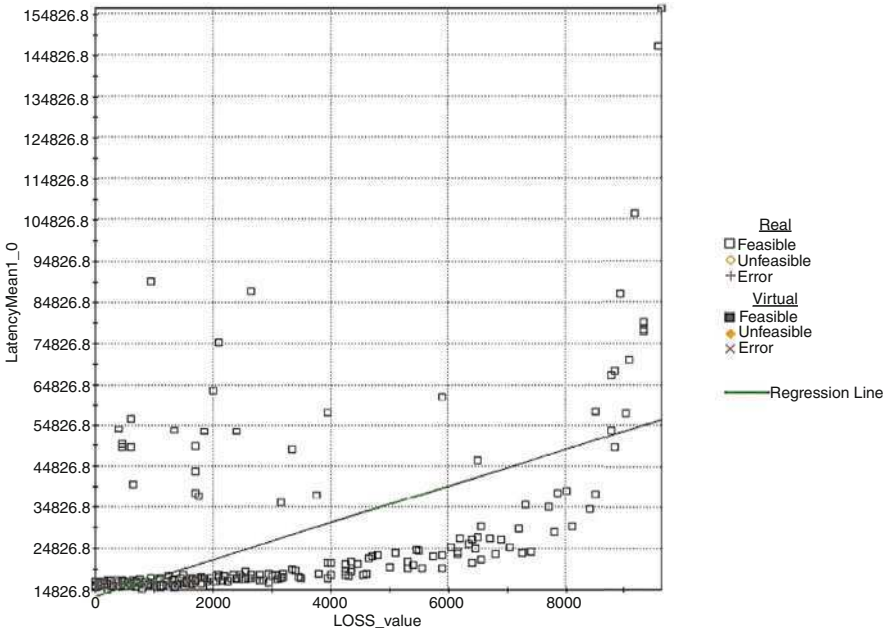

Fig. 7.13 Loss vs Mean Latency, from Node 1 to Node 0

As we can see in Fig. 7.10, for a UDP application flow of 40 Mbps, the physical flow is close to the application flow for low loss rates (with a small overhead). From a loss rate of 20% on, we see that the transmission conditions start to degrade and this is reflected in the fact that the physical throughput has to be increased in order to cope with retransmissions of data. When the loss rate is bigger than 50%–60% even the retransmissions and error corrections are not enough to keep the UDP throughput and packets start to get lost.

In the opposite direction from Node 1 to Node 0 (Fig. 7.11), since the application throughput is very low (around 1 Mbps) the impact of data loss is reduced.

Figure 7.12 shows how the average latency obtained in the communication from Node 0 to Node 1 is kept constant with low loss rates. When loss rate achieves a significant value (35%), we start to see an increasing of the average latency making the multimedia transmission impossible.

In the opposite direction from Node 1 to Node 0, Fig. 7.13 shows that the average measured latency is kept almost constant for low loss rates, becoming a problem as soon as the channel degrades. These results are similar to Fig. 7.12.

As we can see, all the obtained figures are completely coherent with the expected behavior of the system in real deployments based on previous generations of chipsets and the theoretical analysis of the algorithms used. These figures can help us to conclude some interesting data like the fact that the system can have a correct behavior even under poor transmission conditions with a 40% of loss of packet. It is interesting

to note that this result that we have found in a theoretical approach has also been tested on real powerline systems in field trials.

7.2.3.2 Find the Optimum Point for the Design Parameters

Once the correctness of the design has been verified using MULTICUBE procedure, we can proceed to analyze the different results obtained through the automatic simulations in order to find the correct optimum point for the design parameters under study. As we mentioned at the beginning of this study, the objective is to find the correct point where latency is minimized (the transmission of a UDP flow takes minimum time), memory is minimized (since memory is expensive in an embedded system) and throughput is maintained (no application data loss). In order to achieve this ambitious objective, we will go through the database provided by modeFRONTIER and analyze the different figures (Figs. 7.14–7.16).

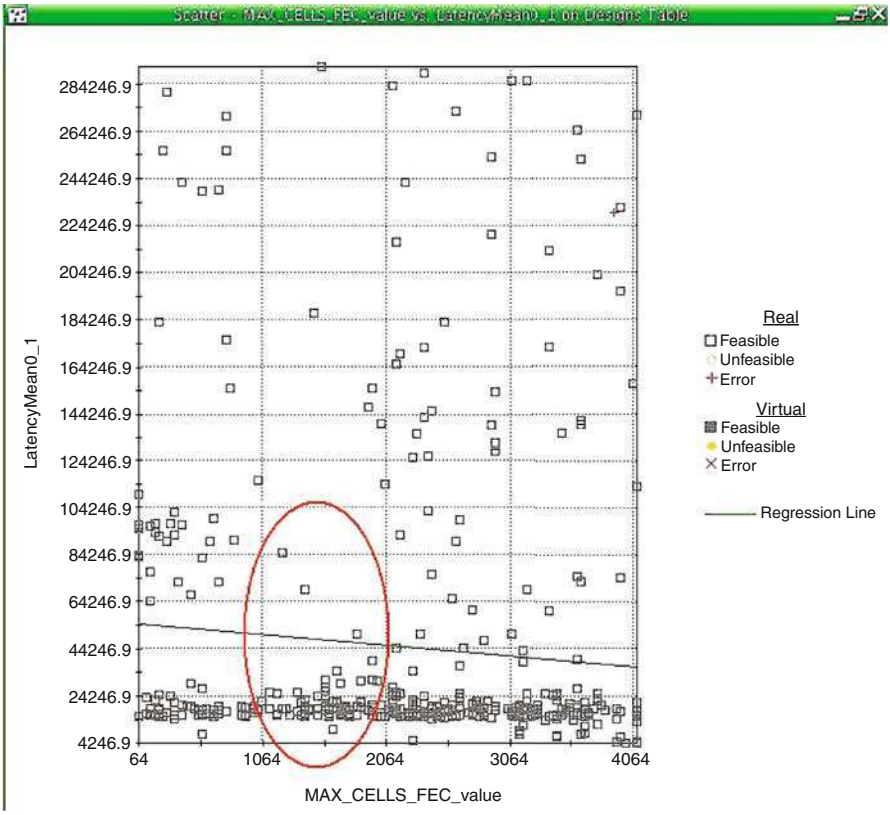

Fig. 7.14 MAX_CELLS_FEC vs Mean Latency, from Node 0 to Node 1

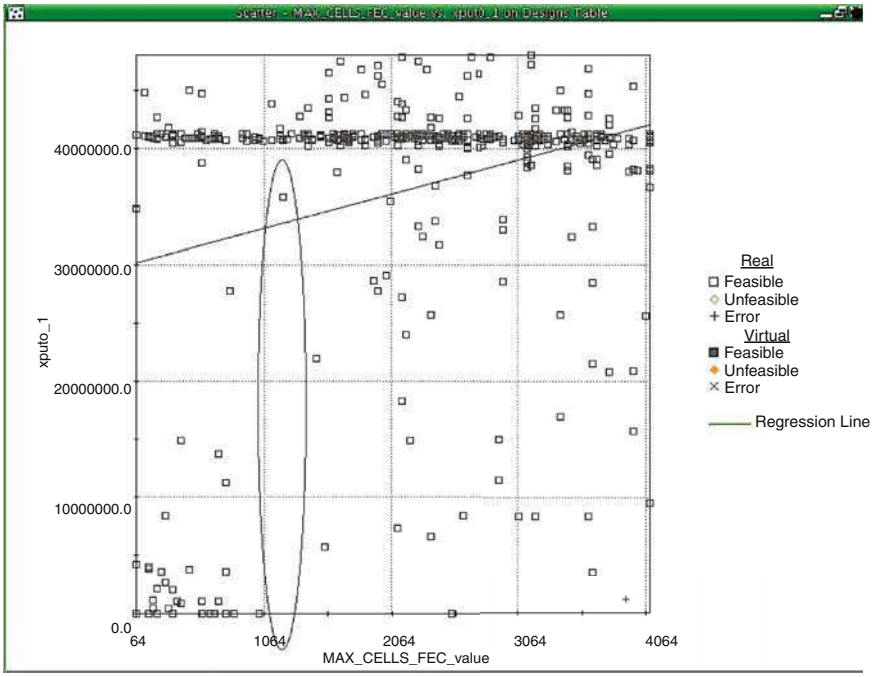

Fig. 7.15 MAX_CELLS_FEC vs Throughput, from Node 0 to Node 1

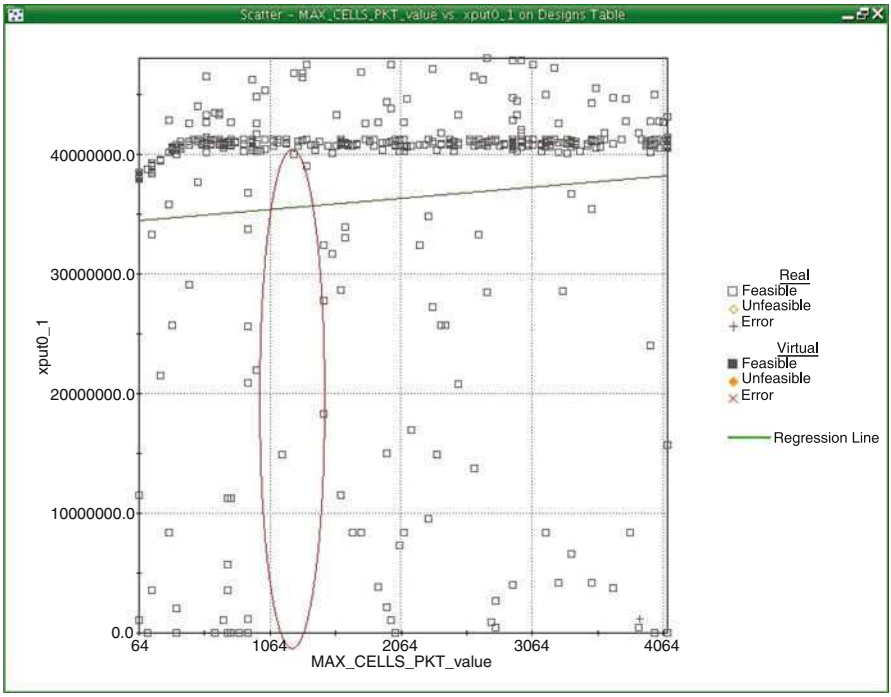

Fig. 7.16 MAX_CELLS_PKT vs Throughput, from Node 0 to Node 1

In all the three figures, we can see that the points that guarantee minimum average latency and minimum memory (translated in number of cells used in packet and FEC memories), while maintaining the pre-set UDP throughput, are reached at around 1,064 cells for PKT and FEC memories. However, a more in depth study with modeFRONTIER analysis tools is needed for deriving the exact figures.

7.2.3.3 Objective Assessment

From an objective point of view, after the assessment of the procedure proposed in MULTICUBE project for system optimization, the overall conclusion is that the proposed flow can save up to a 80% of designer time while achieving better results in terms of performance since much more simulations can be run and analyzed following this flow obtaining 10 times more possible combinations than with the semi-automatic design flow. Also, with just one run we can already identify the optimum for the design parameters. A second or third iteration would fine tune the design in order to obtain the final parameters.

As an example, we have run a test case where we have tried to optimize the parameters described in previous sections and we have measured the values presented Fig. 7.17 (an estimation based on designer's inputs has been made).

As we can clearly see from Fig. 7.17, the automated procedure provides:

- **Higher number of points:** The number of points that can be analyzed by following MULTICUBE methodology is much higher than the ones obtained by using the semi-automated procedure
- **Better quality of the results:** The quality of the results obtained through the use of the automated procedure is higher since modeFRONTIER guarantees the correct

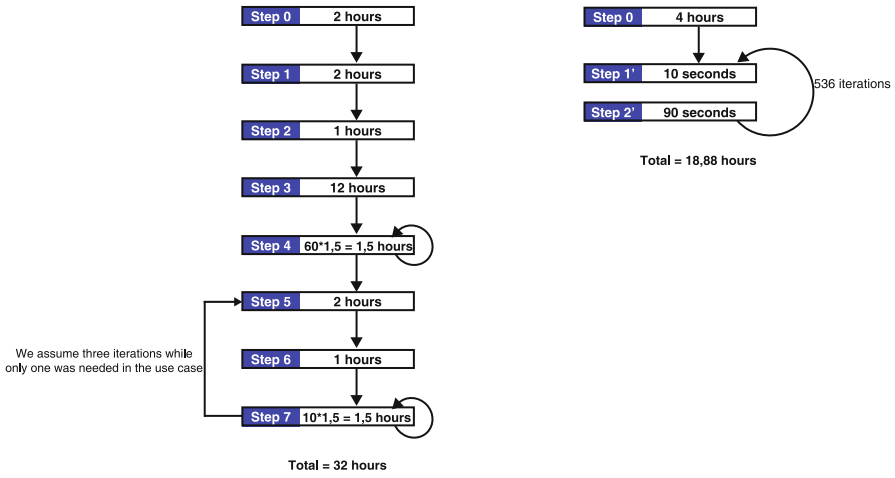

Fig. 7.17 Assessment procedures comparison (semi-automated vs automated)

generation of experiments in order to maximize/minimize the metrics following user guidelines
- **Less complexity of the flow:** The complexity of the flow is much less in the case of the automated procedure. The number of steps to be accomplished is much lower. In this sense, it is interesting to mention that a lower number of steps represents a lower possibility of errors
- **Faster convergence times:** The convergence towards the optimum point is achieved faster with the automated procedure rather than with the semi-automated procedure

7.2.3.4 Subjective Assessment

In parallel with the objective assessment presented in the precedent paragraph, we can mention some subjective facts that have been identified by the engineers that have run the flow in the use case. The main conclusions of this subjective analysis can be:

MULTICUBE design flow is also a perfect tool to guarantee the coherence and correctness of the design as we have seen in our study. The analytical capacities of modeFRONTIER allow to stress the model and to analyze its behavior under different running conditions allowing investigating if the model behaves as expected since we can analyze a larger number of operating conditions. As a simulator tool alone, it adds value to the current simulation methodologies in place.

We can mention that the procedure proposed by MULTICUBE makes the work of the embedded designer much easier by proposing an excellent front-end tool and by automating all the processes necessary to update the platform in each simulation cycle. The intervention of the designer is reduced to a minimum, minimizing also the possibility of human errors.

MULTICUBE design flow can also benefit from the use of standardized interfaces. It is possible then to envision the usage of other analysis tools quite easily. In addition to powerful analysis tools from commercial suites like modeFRONTIER, that have been proven in other technical domains, we can use other suites developed internally in the company or to adapt it to future tools.

7.3 Conclusions

This chapter detailed the use of MULTICUBE design flow in the powerline multimedia transmission use case for system optimization. It shows that the proposed flow can save up to a 80% of designer time while achieving better results in terms of performance since much more simulations (up to ten times more possible combinations than with the semi-automatic design flow) can be run and analyzed. Also, with just one run we can already identify the optimum for the design parameters. A second or third iteration would fine tune the design in order to obtain the final parameters.

In parallel, MULTICUBE design flow is also a perfect tool to guarantee the coherence and correctness of the design as we have seen in our study.

Finally, from a subjective point of view, we can mention that the MULTICUBE design flow makes the work of the embedded designer much easier by proposing an excellent front-end tool and by automating all the processes necessary to update the platform in each simulation cycle. Also, the designer can benefit from the powerful analysis tools from commercial suites like modeFRONTIER, that have been proven in other technical domains, in order to select the optimum solution.

The overall conclusion of the utilization of MULTICUBE design flow over a real industrial use case discussed in this chapter is that the added value overcomes largely the cost of setting up the whole process, converging much faster to the optimum solution for a given technical problem.

Chapter 8
Design Space Exploration of Parallel Architectures

Carlos Kavka, Luka Onesti, Enrico Rigoni, Alessandro Turco, Sara Bocchio, Fabrizio Castro, Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, Giovanni Mariani, Fan Dongrui, Zhang Hao, and Tang Shibin

Abstract This chapter will present two significant applications of the MULTICUBE design space exploration framework. The first part will present the design space exploration of a low power processor developed by STMicroelectronics by using the modeFRONTIER tool to demonstrate the DSE benefits not only in terms of objective quality, but also in terms of impact on the design process within the corporate environment. The second part will describe the application of RSM models developed within MULTICUBE to a tiled, multiple-instruction, many-core architecture developed by ICT, Chinese Academy of Sciences, China. Overall, the results have showed that different models can present a trade-off of accuracy versus computational effort. In fact, throughout the evaluation, we observed that high accuracy models require high computational time (for both model construction time and prediction time); vice-versa low model construction and prediction time has led to low accuracy.

8.1 Introduction

The embedded systems industry faces today an always increasing demand to handle complexity in the design process, which together with the usual strong time-to-market pressure, make essential the use of an automatic tool-based design flow. The use of an automatic tool for Design Space Exploration (DSE) directly impacts on the entire design process within the corporate, with benefits that can be measurable (or tangible), like the reduction of the overall design process lead time, and also qualitative (or intangible) like the streamlining and the reduction of human error prone repetitive operations.

DSE tools for the automation of the embedded system design process do exist today. The MULTICUBE FP7 European Research project has contributed to the

C. Kavka (✉)
ESTECO, Trieste, Italy
e-mail: carlos.kavka@esteco.com

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_8, © Springer Science+Business Media, LLC 2011

development of an Open Source tool and to the re-targeting of a proprietary state-of-the-art multi-disciplinary optimization tool that can be directly applied today to embedded systems design. One of these tools is M3Explorer [6], an open source tool developed from scratch in the MULTICUBE project, and the other is modeFRONTIER™ [2], an already existing proprietary tool widely used in multidisciplinary optimization, which has been re-targeted to the domain of embedded systems.

This chapter presents two case studies of design space exploration in embedded systems to illustrate the benefits of the use of automatic tools from an industrial perspective. Section 8.2 describes a design space exploration study performed on a low power processor design and Sect. 8.3 describes the application of response surface models on a many-core architecture design.

8.2 Design Case Study: Design Space Exploration of the STM Industrial SP2 Platform

This section describes the application of automatic design space exploration for the design of a low-power processor developed by STMicroelectronics, using the modeFRONTIER multidisciplinary optimization tool. The objective is to demonstrate the benefits of the introduction of an automatic design process not only by considering the final objective quality, but also its effects on the entire design process within the corporate. The experiment is an extension of the benchmark used in Chap. 3 to analyze the behavior of the optimization algorithms proposed in the MULTICUBE project [7].

8.2.1 Architectural Model Description

The SP2 processor from STMicroelectronics is a low power and high performance microprocessor offering comparable performance to entry-level desktop microprocessors. It is designed for both generic and mobile applications that need low power dissipation and high peak performance. The presence of a SIMD coprocessor makes it suitable for multimedia applications.

The main architectural features of the SP2 processor can be summarized as follows:

- MIPSISA32 release2 compatible instruction set architecture (ISA) and privilege resource architecture (PRA)
- 4-issue superscalar out-of-order execution
- Built-in 2 integer ALUs, 2 interleaved load-store units
- Split primary instruction and data cache
- 64-byte cache line, 4-way set associative primary caches
- 64-byte cache line, 8-way set associative unified exclusive secondary cache

- Configurable out-of-order parameters: renaming register number, instruction issue window width, reorder buffer depth, etc
- Configurable cache parameters: cache size
- Runtime resizable secondary cache

The experiments presented in this section were performed using Sp2sim, a register transfer level cycle accurate simulator, which runs its applications according to SP2's pipelines and configurations. Sp2sim models precisely the SP2 microprocessor design, which includes both the MIPSISA32r2 ISA and PRA [5]. It also embeds a model of external memory controller, which can be configured to measure the performance for different memory latencies and sizes. The interface between the processor model and the memory controller model is a 64-bit AXI interface. Sp2sim implements both an area and power consumption estimation algorithms. Sp2sim is written in C++ language so as to achieve high simulation speed. It runs on Linux x86 or x86_64 platforms.

8.2.2 Design Space and Application

The SP2 implementation provides eleven configurable parameters, which are classified into three categories: out-of-order execution engine, cache system and branch prediction. By adjusting these parameters, the user can get different processor implementations targeting to distinct application areas. The list of parameters together with a description and their possible values is presented in Table 8.1.

The SP2 simulator produces seven system metrics, which are grouped in three categories: performance, power dissipation and area occupation. Table 8.2 describes the system metrics.

The selected application for the experiments presented in this section is *gzip*, a popular data compression program written by Jean-Loup Gailly, which comes from SPEC CPU2000 benchmark suite [8]. It has been selected since its size is relatively small, and the program behavior is more regular than many other applications.

Table 8.1 The eleven configuration parameters

Category	Parameter	Description	Values
Out of order execution	rob_depth	Reorder buffer depth	32, 48, 64, 80, 96, 112, 128
	rmreg_cnt	Rename register number	16, 32, 48, 64
	iw_width	Instruction window width	4, 8, 16, 24, 32
Cache system	icache_size	Instruction cache size	16, 32, 64
	dcache_size	Data cache size	16, 32, 64
	scache_size	Secondary cache size	0, 256, 512, 1024
	lq_size	Load queue size	16, 24, 32
	sq_size	Store queue size	16, 24, 32
	msh_size	Miss holding register size	4, 8
Branch prediction	bht_size	Branch history table size	512, 1024, 2048, 4096
	btb_size	Branch target buffer size	16, 32, 64, 128

Table 8.2 The system metrics generated by the simulator

Category	Metric	Description
Performance	total_cycle	Total cycle number
	total_inst	Total instruction number
	ipc	Instruction per cycle
Power dissipation	total_energy	Total energy consumed
	power_dissipation	Average power dissipation
	peak_power_dissipation	Peak power dissipation
Area	area	Area occupied

8.2.3 Design Space Exploration

The aim of the experiments presented in this section is to determine the set of designs that minimize the total number of cycles, the power dissipation and the area occupation required to run the selected *gzip* application with the SP2 simulator. Since it is a multi-objective problem, and since the three objectives can potentially be contradictory, the result of the optimization process will not be a single design, but a Pareto front, which corresponds to the set of designs that represents a trade-off between the different objectives.

In real industrial applications, one of the hardest constraints that limits the exploration is determined by the computing resources available for simulation. In this study, the complete design space consists of 1,161,216 designs by considering all combinations of the configuration parameters. This space is clearly too large to be explored exhaustively, making essential to perform a well defined exploration strategy.

In this experiment, the time available for simulation limits the exploration to the execution of at most 8,134 evaluations of the simulator. This information will guide the selection of the algorithms for the different exploration phases. The rest of this section illustrates the design space exploration process, starting with the creation of the optimization workflow, the definition of the experiments, the optimization process and the results assessment.

8.2.3.1 The Optimization Workflow

The optimization tool used in this experiments is modeFRONTIER, a multidisciplinary multi-objective optimization and design tool which is used world-wide in many application fields like aerospace, appliances, pharmaceutics, civil engineering, manufacturing, marine multibody design, crash, structural, vibro-acoustics and turbo-machinery. In the FP7 MULTICUBE project, modeFRONTIER has been enhanced to support categorical discrete domain optimization problems, like typical problems faced in the SoC domain.

The design space exploration process starts with the definition of the optimization workflow, which is presented in Fig. 8.1. The workflow can be graphically defined or can be built automatically using the XML design space definition file (see Chap. 1).

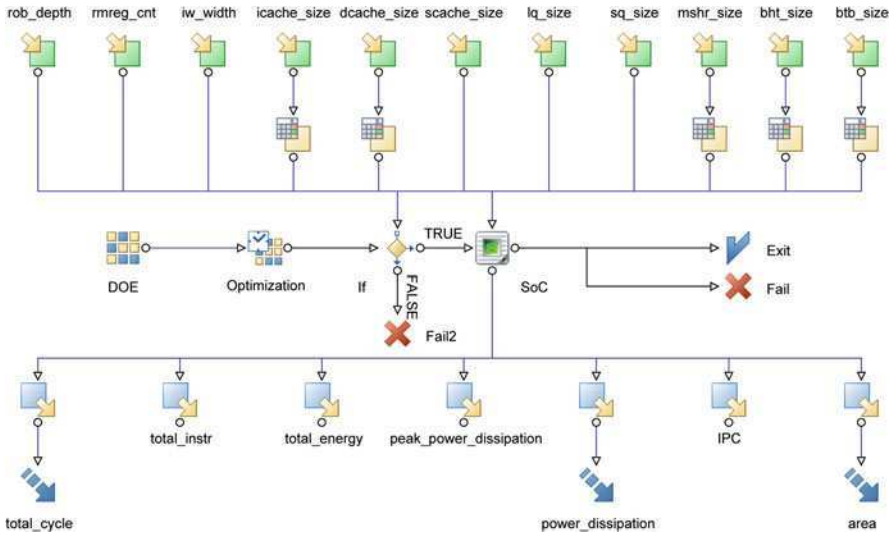

Fig. 8.1 The modeFRONTIER optimization workflow

The worklow specifies both a data path and a process path, which represent respectively, the data flow for the evaluation of a design and the optimization execution flow as described below.

The data path, which flows vertically from the top to the bottom, contains one input node for each configuration parameter and one output node for each metric produced by the simulator. In this optimization task there are eleven input nodes and seven output nodes. The System-on-Chip node (labeled as SoC) represents the interaction with the simulator, and takes care of the interface between the exploration tool and the simulator, passing values for the configuration parameters and getting back the values of the metrics as produced by the simulator. The three objectives to be minimized are represented with the arrow nodes.

The process path, which flows horizontally from left to right, starts with the Design of Experiments node (DoE), which generates the initial set of configurations to be evaluated. The optimization node (Optimization) guides the exploration of the design space using an optimization algorithm to generate the subsequent configurations that will be evaluated based on the performance of previous evaluations. The conditional node (If) passes only feasible designs for evaluation to the System-on-Chip node. Successful executions of the simulator will generate a valid design (Exit), and errors will generate failed designs (Fail).

8.2.3.2 Design of Experiments

The Design of Experiments (DoE) technique is used to generate the initial set of designs for evaluation. The aim is to define a limited number of test runs that allow to maximize the knowledge gained by eliminating redundant observations, in such a

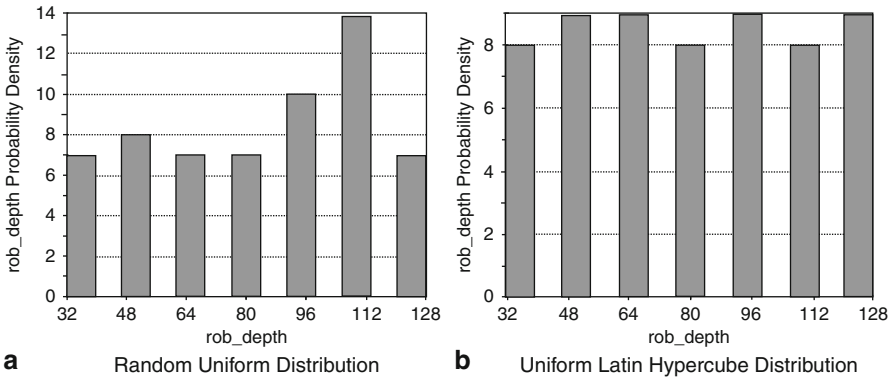

Fig. 8.2 The marginal distribution for the parameter rob_depth with a **a** "Uniform Random" and **b** "Uniform Latin Hypercube" Design of Experiments with 60 designs. It can be appreciated that the distribution of samples for each value is better in **b**, showing that the second approach has provided a better sampling of this discrete design space

way, that the time and resources needed to make the experiments is reduced as much as possible.

In these experiments, a Uniform Latin-Hypercube algorithm [4] is selected to generate 60 initial designs. The selection of the DoE algorithm and the number of designs are driven by the fact that only 8,134 designs can be evaluated and as described in Chap. 3, this algorithm is preferred over the widely-used Uniform Random algorithm, particularly due to the better mapping of the marginal distribution functions with a small number of samples, as clearly shown in Fig. 8.2.

8.2.3.3 Optimization

The algorithm MFGA (Magnifying Front Genetic Algorithm) [7], described in Chap. 3, has been selected by considering the characteristics of the optimization problem. This algorithm has been developed in the MULTICUBE project to handle categorical discrete optimization problems. A non-generational operation mode (steady state) makes it well suited for problems involving long simulation time. This algorithm has only two configuration parameters, which are set to default suggested values: crossover probability to 0.9 and mutation probability to 0.15.

The optimization is performed by using this algorithm till the allowed number of evaluations is reached without any manual intervention. Figure 8.3 shows the design space evaluation, highlighting the Pareto front as obtained at the end of the evolutionary process[1]. The values of the metrics (normalized due to confidentiality reasons) are plotted for each evaluated design with a bubble which size is directly

[1] Even if unusual, a bubble graph with the third dimension as the bubble size was preferred over a typical 3D graph since the large number of points produces a difficult to understand 3D cloud-like graph

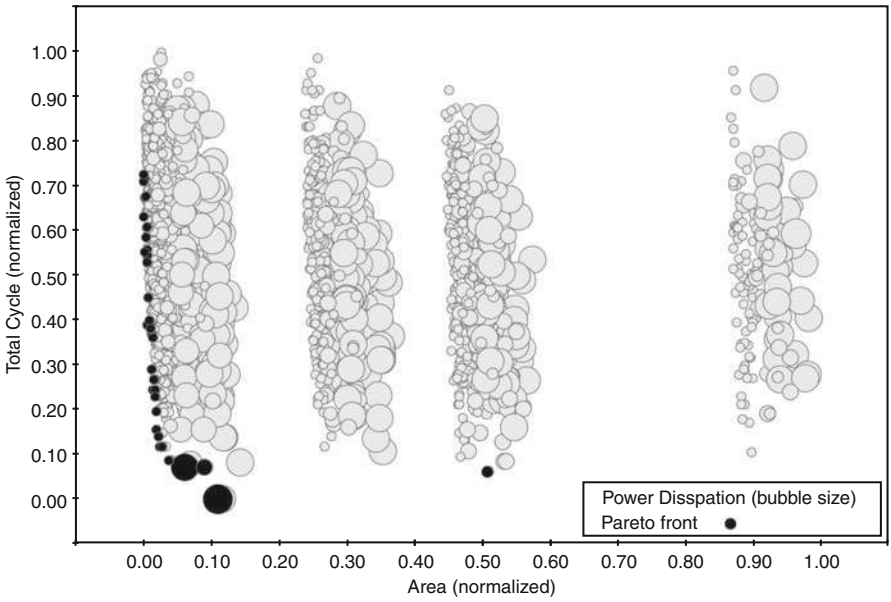

Fig. 8.3 The Pareto front in the design space

proportional to the Power Dissipation. The x and y axis correspond respectively to the Area and the Total Cycle. Note that designs with lower Occupation Area, lower Total Cycle and lower Power Dissipation are preferred. The set of Pareto designs is represented with black bubbles.

Figure 8.4 presents a comparison between the Pareto front obtained in this experiment and the Pareto front obtained by a semi-automatic approach guided by statistical analysis, which was performed as described in Sect. 3.4.2. As the figure clearly shows, the Pareto front obtained by the fully automatic procedure dominates all points generated by the semi-automatic approach. This is an important result since the automatic procedure was performed without any manual intervention, while the semi-automatic procedure was based on the designer ability and experience to assess the results and based on that, select the next instance of the model to be simulated.

Table 8.3 presents some quantitative results of the comparison between the Pareto fronts obtained with the automatic optimization and the semi-automatic approach. The enhancement on Area and Total Cycle metrics are displayed in percentage of the whole design space grouped by their power dissipation characteristics. It can be appreciated that more than interesting enhancements have been obtained by the automatic procedure.

8.2.3.4 Other Studies

Once a well organized experiment provides enough data, it is possible to study some properties of the design space. One of the most important analysis is the correlation

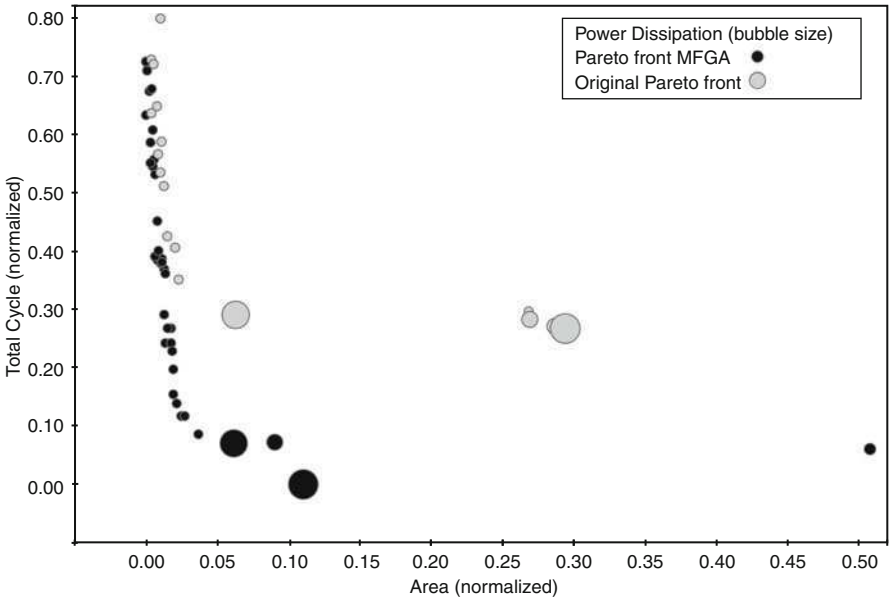

Fig. 8.4 A comparison between the Pareto front obtained with the automatic optimization and the semi-automatic process based on statistical analysis

Table 8.3 Enhancement on Area and Total cycle metrics when compared with the statistically-based approach grouping designs by Power dissipation

Power dissipation	Area	Total cycle %
Low power dissipation	Comparable	−7.71
Medium power dissipation	−17.35%	−18.42
High power dissipation	Comparable	−28.99

study, which allows to identify relations between parameters and metrics. There are many tools that can be used, however, the Self Organizing Map (SOM) [3] is particularly well suited for high dimensional spaces. Its main advantage compared with other methods is that it allows to identify not only the global correlations between the variables, but also highlights information about local correlations.The SOM is a 2D map where the parameters and metrics are automatically mapped during the learning process. Figure 8.5 shows the SOM obtained after the optimization process described in the previous section.

Variables (parameters and metrics) that are correlated are mapped to nearby areas of the map. The spatial location of the variables total energy (total_energy), power dissipation (power_dissipation) and data cache size (dcache_size), which are all mapped in the lower-right corner of the map, allows to conclude that there is a strong correlation between them. There is also a large correlation between the occupation area (area) and the secondary cache size (scache_size), and between total cycle (total_cycle) and store queue size (sq_size).

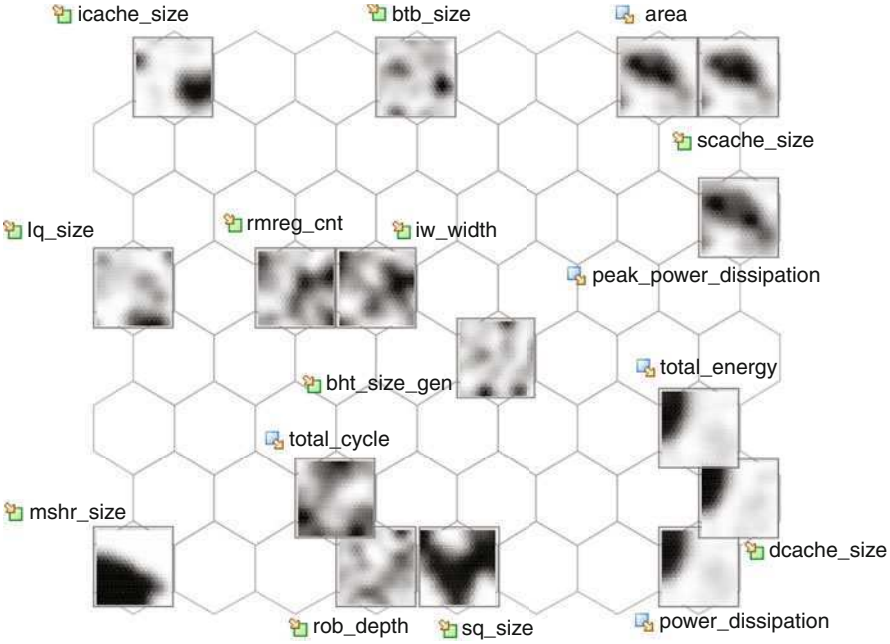

Fig. 8.5 The Self Organizing Map describing correlations between parameters and metrics. Some low correlations parameters and metrics have been removed from the diagram to enhance readability

The SOM provides also more information: the gray-scale patterns indicate the locations where the values of the variables are mapped, with light gray corresponding to low values and dark gray corresponding to large values. Just as an example, the darker area in the upper-left corner of the pattern of total energy (total_energy) indicates that higher values of this variable were mapped in the upper-left corner of the map. In this way, similar patterns in grouped variables describe a direct correlation, while inverted color patterns describe an inverse correlation. The analysis suggests there is a direct correlation between occupation area (area) and static cache size (scache_size), while there is an inverse correlation between total cycle (total_cycle) and store queue size (sq_size). Local correlations can also be identified by studying particular areas of the gray-scale patterns.

8.2.4 Conclusions

The automatic optimization process allowed to obtain a better Pareto front when compared with equivalent experiments performed by a semi-automatic approach guided by a statistical analysis. Not only the Pareto front was better at the end of the optimization, but after the evaluation of only 3,000 designs, the obtained Pareto

front was better than the front obtained by the statistical analysis guided optimization which consisted of 14,216 evaluations (see Chap. 3).

There are other advantages of the automatic exploration procedure. In the automatic exploration, all data concerning previous evaluations are always stored in a structured database. The designer, not only will not be stuck on repetitive operations, but can focus his/her attention (and profit from his/her experience) in analyzing the designs database in a statistical manner. In this way, analysis like clustering or correlations can be performed in a more systematic and organized way.

Moreover, the automatic exploration is driven by an optimization engine based on several optimization algorithms, whereas the manual exploration is based on designer ability and experience to assess the results and to move towards the next instance of the model to be simulated. In that particular case (see Chap. 3), wrong assumptions on the statistical distributions of data reduced the possibility to obtain better solutions.

8.3 Design Case Study: Analysis of Performance and Accuracy of Response Surface Models for the Many-Core Architecture Provided by ICT

This section describes an example application of response surface models on a many-core architecture by describing how the models are selected, trained and validated to provide the best accuracy without sacrificing performance.

8.3.1 Platform Description

The ICT many-core (named *Transformer*) is a tiled, multiple-instruction, multiple-data (MIMD) machine consisting of a 2D grid of homogeneous, general-purpose compute elements, called cores or tiles. Instead of using buses or rings to connect the many on-chip cores, the transformer architecture combines its processors using 2D mesh networks, which provide the transport medium for off-chip memory access and other communication activity.

By using mesh networks, the Transformer architecture can support anywhere from a few to many processors without modifications to the communication fabric. In fact, the amount of in-core (tile) communications infrastructure remains constant as the number of cores grows. Although the in-core resources do not grow as tiles are added, the bandwidth connecting the cores grows with the number of cores.

As Fig. 8.6 shows, the Transformer Processor Architecture consists of a configurable 2D grid of identical compute elements, called nodes. Each node is a powerful computing system that can independently run an entire application. The perimeters of the mesh networks in a Transformer Processor connect to memory controllers, which in turn connect to the respective off-chip DRAMs through the chips pins. Each node combines a processor and its associated cache hierarchy with a router,

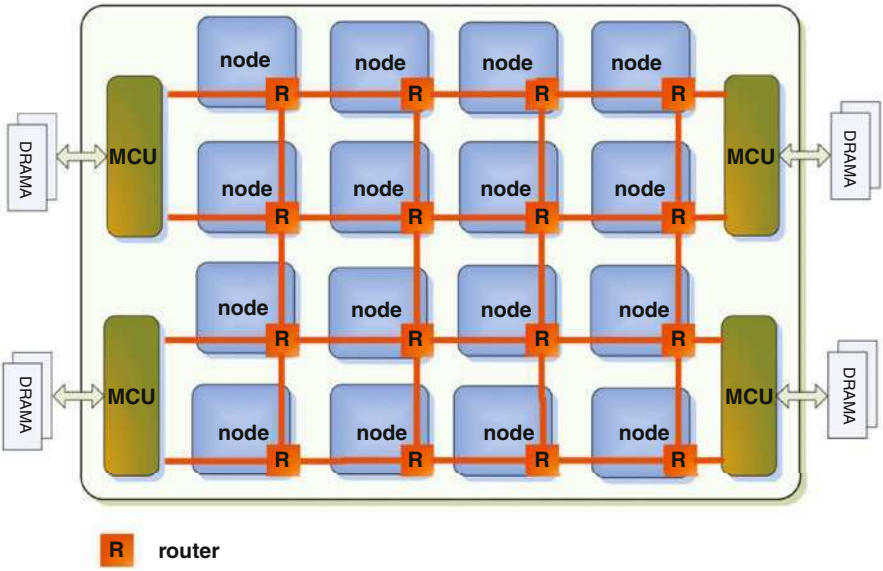

Fig. 8.6 The ICT Many-Core architecture (*transformer*)

which implements the interconnection network. Specifically, each node implements a two-issue out-of-order pipelined processor architecture with an independent program counter and a two-level cache hierarchy. The memory sub-system is *distributed* (thus not shared among cores) to simplify the architecture of the system.

8.3.2 Node Architecture and Instruction Set

For each node there is a separate level 1 instruction/data cache and a unified level 2 cache in each node. As there is need for 64-bit computing in multimedia applications, 64-bit MIPS-III ISA is chosen for the node. Some MAC (Multiply-Accumulate) instructions which need three source operands are also supported by node.

Each node implements a two-issue out-of-order pipelined processor architecture, as shown in Fig. 8.7. The pipeline is divided into 6 stages: fetch, decode, map, issue, execute and write back. For memory operations, there is one more stage *data cache* between *execute* and *write back*. In order to achieve reasonable performance and implement a low complexity node, a *scoreboard*-based out-of-order pipeline based is used. The scoreboard unit in each node is responsible for accepting decoded instructions from the *map* stage, and issuing them to the functional units (address generators, ALUs and FPUs) satisfying dependencies among the instructions. To achieve this goal the main element of the scoreboard is the instruction queue which holds decoded instructions and issues them once the resources they require are free and their dependencies have been cleared.

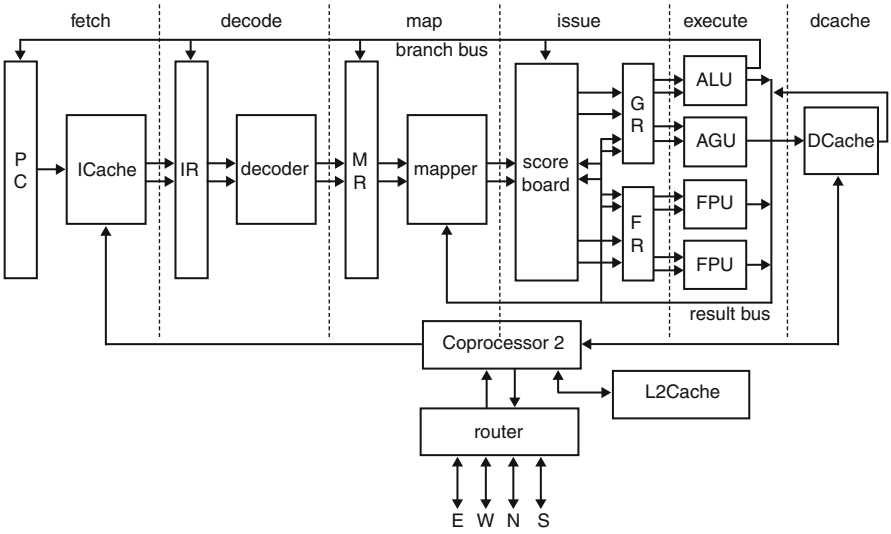

Fig. 8.7 The ICT Many-Core pipelined architecture of a node

8.3.3 The Simulator

The NoC based many-core design transformer is implemented in C++ as a cycle-accurate simulator. The simulator accepts the configuration as an input file, and generates the system metrics as an output file. Both input and output files are defined in XML format by following the rules described in the Design Space Definition file, as described in Chap. 1.

Figure 8.8 shows the scheme of a single core. A core defines an instruction queue, whose entry is allocated to the newly instructions fetched from instruction cache, and released after the instruction was committed. In detail, an instruction fetched from instruction cache (1) is allocated in the instruction queue (2), and at the same time, the instruction dependency is built (3). Then, if the required logic unit is idle and the necessary dependency is satisfied, the instruction is issued (4) to the corresponding logic unit, and the state of the logic unit is changed to *busy* in order to block the instructions using it (5). After the predefined latency and essential operations (e.g. accessing the register file, accessing the local data cache, and routing message in the mesh), the instruction is committed, and simultaneously those interrelated resources and dependencies are released. In above mentioned process, the timing information is accurately collected to implement a cycle-accurate simulator.

The power model is based on the Princeton University's Wattch power model [1], with all parameters updated accordingly to a 0.13 μm process.

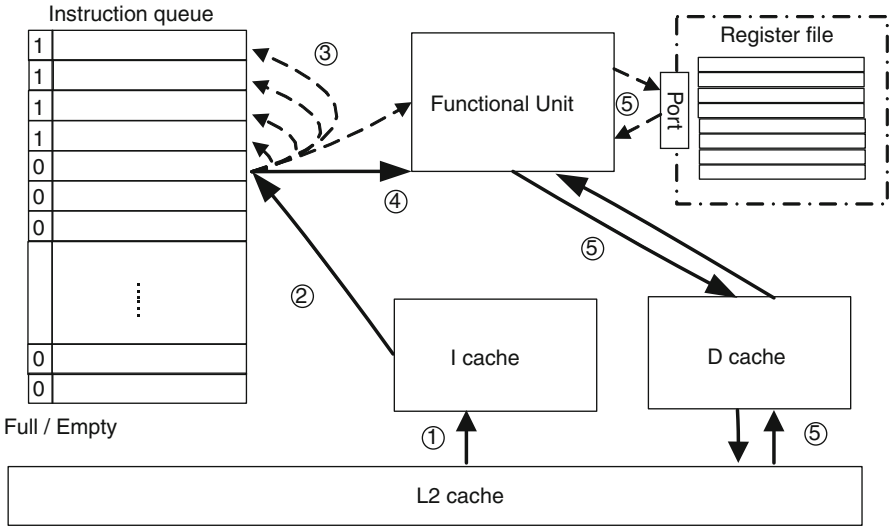

Fig. 8.8 The ICT Cycle-Accurate Simulator

8.3.4 Design Space and Application

One goal of the design of Transformer is to make a reconfigurable platform, allowing further tuning as the evaluation works goes on. According to the system metrics, the designers can tune the hardware arguments for higher performance. In the simulator, all the tunable arguments are collected in a configuration file in XML style. When the simulator is started, it reads the arguments from this configuration file. The design space, which is shown in Table 8.4, is composed of 1,134 design points. As an additional rule in the design space, the associativity of the level-2 cache should be greater than the sum of the associativities of the level 1 instruction and

Table 8.4 Parameter space for the ICT many-core platform

Parameter	Minimum	Maximum
Mesh_order	2	8
Cache_block_size	32	64
ICache_ways	2	16
Icache_entries	128	512
DCache_ways	2	16
DCache_entries	128	512
L2Cache_ways	4	32
L2Cache_entries	128	512
L2Cache_access_latency	3	10
Memory_size	8	32
Memory_access_latency	30	100
Router_buffer_entries	2	8

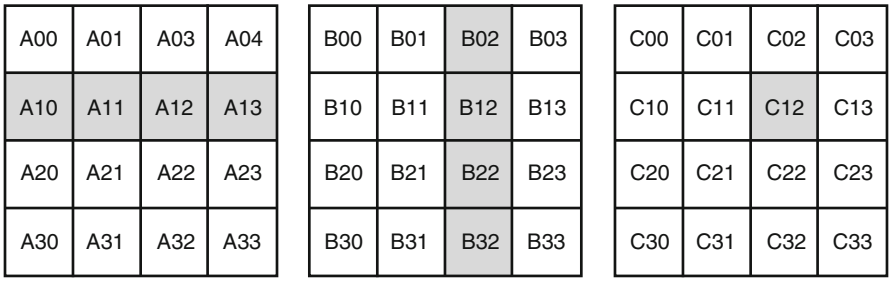

Fig. 8.9 Parallel multiplication of matrices

data caches. The system metrics associated with the architecture are the following: cycles, instructions, power dissipation, peak power dissipation and area.

Application The application used for the following experimental results is a classic implementation of *partitioned* matrix multiplication. Each matrix (both sources and destination) is divided into an equal number of sub-matrices; each node works by multiplying the rows and columns needed for a single partition of the destination matrix. Figure 8.9 shows an example matrix multiplication ($C = A \times B$) for two squared matrices divided into 16 sub-matrices ($A_{i,j}, B_{i,j}, C_{i,j}$ where $i, j \in \{0, 3\}$).

8.3.5 Response Surface Modeling of Many-Cores

This section presents the results of the validation of RSMs described in Chap. 4. For the sake of synthesis, in this section the results related to the following RSM configurations are reported:

- *Linear regression*
  - Model order: *first*,
  - *Without any interaction between parameters*,
  - *Excluding* the following design space parameters from metric estimation:
    - *ICache_ways*,
    - *DCache_ways*,
    - *L2Cache_ways*,
    - *L2Cache_access_latency*,
    - *Memory_size*,
    - *Memory_access_latency*.
- *Radial Basis Functions*
  - Distance function definition: *thin plate spline*.
- *Splines*
  - No parameters.
- *Kriging*

  - Variogram Type: *Gaussian*,
  - Autofitting Type: *Maximizing Likelihood*,
- *Evolutionary Design*
  - Crossover Depth: *10*,
  - Generations Number *1000*,
  - Population Size: *500*,
- *Neural Networks*
  - Network Size Policy: *Automatic*.

To enable the validation of the proposed RSMs, all the simulations associated with the complete design space (1,134 design points) have been performed. The resulting data have been used to train the RSMs and to validate the predictions. The chosen sizes for the training sets are: 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1,000 design space points. For each training set size, 5 training sets have been generated with a pseudo-random process from the simulation data, hence the total number of training sets is 50. Given a trained RSM, the average normalized error of the predictions over the complete design space (of 1,134 points) is computed. For each RSM, the three values of the Box-Cox transform $\lambda = \{0, 0.5, 1\}$ were evaluated. Overall, for each RSM, 150 validation tests were performed.

Figure 8.10 reports ε versus the training set size, for all the experimented RSMs, where:

$$\varepsilon = \frac{\eta_1 + \eta_2 + \eta_3 + \eta_4 + \eta_5}{5} \tag{8.1}$$

and η_i, with $1 \leq i \leq 5$, is the *average normalized error* corresponding to the error observed training the RSM with the $i - th$ training set of the corresponding size. In Fig. 8.10, ε is the *average normalized error* on the y-axis, while the x-axis shows the training set size.

Linear Regression, Radial Basis Functions and Splines have been plotted with a scale from 0 to 0.7 (equivalent to 70% error); Kriging, Neural Network and Evolutionary Design have a scale ranging from 0 to 0.045 (equivalent to 4.5% error).

Overall, there is evidence that the Neural Network (Fig. 8.10f) allows for the best error for a given training set size (less than 0.2% error after 100 training samples). On the other hand, the Linear Regression RSM seems to be not appropriate for modeling such type of use case (Fig. 8.10a) since it provides the highest error while not scaling with the training set size.

As can be seen, the logarithmic box-cox transformation is crucial for reducing the variance of the data and improves the model prediction. We further analyze the behavior for this particular Box-Cox transform in Fig. 8.11. The Figure shows the statistical behavior (computed over 5 runs on the same training set size for each of the RSMs). As can be seen, Evolutionary Design, Kriging and Neural Network provide a strong improvement with respect to Linear Regression and Splines (their scale has been reformatted accordingly), while Radial Basis Functions are overall halfway.

However, for the ICT use case, Evolutionary Design and Neural Network have revealed the highest execution times (the validation for Evolutionary Design lasted for

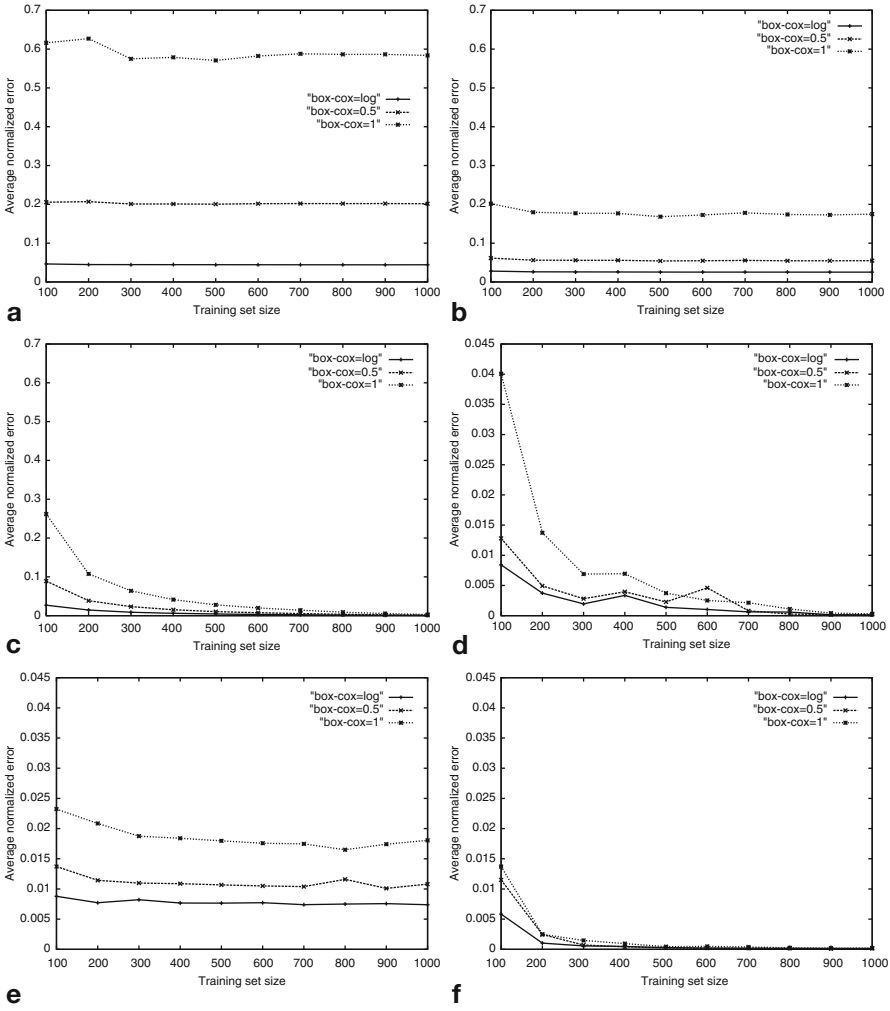

Fig. 8.10 Average normalized error versus training set size for the experimented RSMs. **a** Linear regression, **b** Splines, **c** Radial basis functions, **d** Kriging, **e** Evolutionary design, **f** Neural networks

several days, while Neural Network took few hours), Splines and Linear regression have been the fastest RSMs (the validation time for both of them was few minutes), while Radial Basis Functions and Kriging presented an intermediate behavior (the overall validation time was several minutes).

Overall, the results have showed that different models can present a trade-off of accuracy versus computational effort. In fact, throughout the evaluation, we observed that high accuracy models require high computational time (for both model construction time and prediction time); vice-versa low model construction and prediction time has led to low accuracy. We can sum up by observing that the best choice

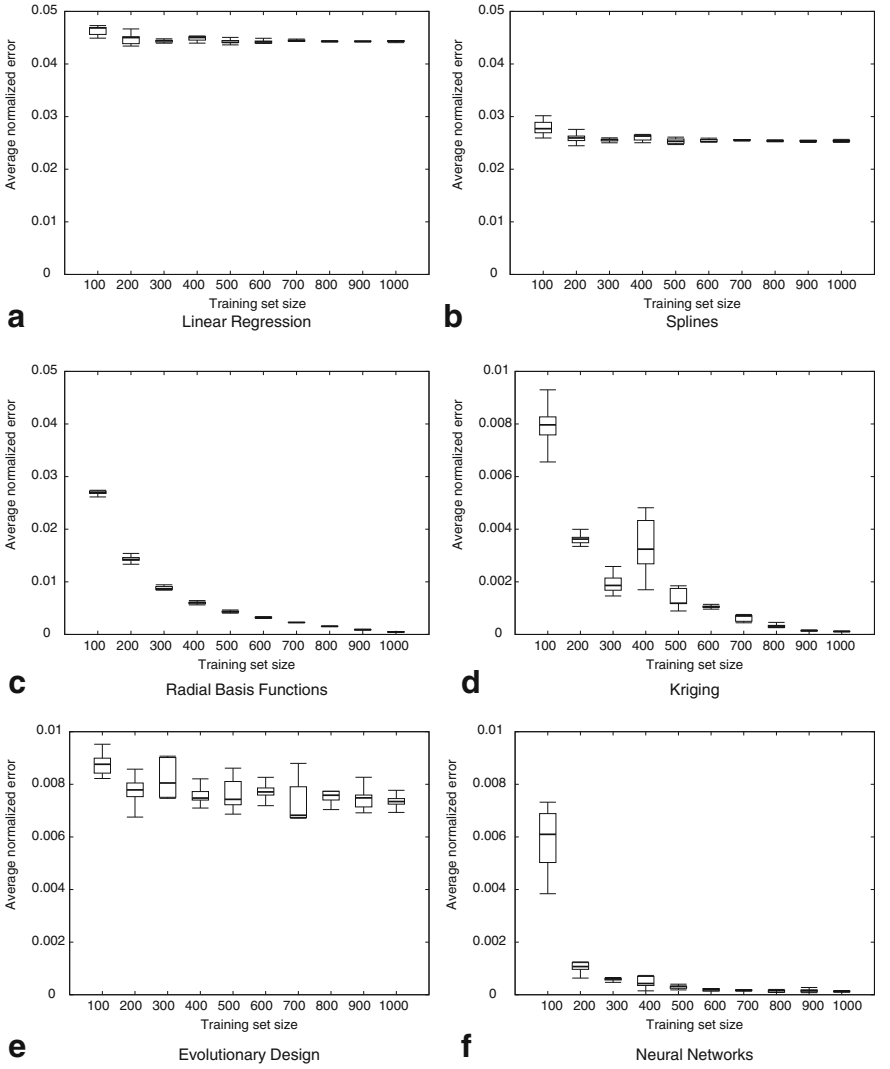

Fig. 8.11 Box plots of average normalized error versus training set size for logarithmic box-cox transformation. **a** Linear regression, **b** Splines, **c** Radial basis functions, **d** Kriging, **e** Evolutionary design, **f** Neural networks

for a specific use case depends on the desired accuracy and the constraints on the prediction time. Referring to the use-case addressed in this section, however, we discovered an exception to the previous *empiric* rule when, for $\lambda = 0$, Kriging gave a better accuracy than Evolutionary Design for every training set size, paired with a lower computation time.

8.4 Conclusions

This chapter presented two significant applications of the MULTICUBE design space exploration framework. While in the first part has been presented the design space exploration of a low power processor developed by STMicroelectronics by using the modeFRONTIER tool, the second part has described the application of RSM models developed within MULTICUBE to a tiled, multiple-instruction, many-core architecture developed by ICT China. The presented analysis refers to a direct application of the methodology implemented in the project to real case studies.

References

1. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations. In: ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture, pp. 83–94. ACM, New York, NY, USA (2000)
2. ESTECO: modeFRONTIER, Multi-Objective Optimization and Design Environment Software. http://www.esteco.com
3. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics **43**(1), 59–69 (1982)
4. McKay M. D., C.W.J., J., B.R.: Latin hypercube sampling: A comparison of three methods for selection values of input variables in the analysis of output from a computer code. Technometrics **22**(2), 239–245 (1979)
5. MIPS Technologies: Architecture Programming Publications for MIPS32. http://www.mips.com/products/product-materials/processor/mips-architecture
6. MULTICUBE: Final Prototype of the Open-Source MULTICUBE Exploration Framework. Deliverable 3.1.2 (2009)
7. Turco, A., Kavka, C.: MFGA: a GA for complex real-world optimisation problems. International Journal of Innovative Computing and Applications **3**(1), 31–41 (2011)
8. Standard Performance Evaluation Corporation: SPEC CPU 2000. http://www.spec.org/cpu2000

Chapter 9
Design Space Exploration for Run-Time Management of a Reconfigurable System for Video Streaming

Giovanni Mariani, Chantal Ykman-Couvreur, Prabhat Avasare, Geert Vanmeerbeeck, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria

Abstract This Chapter reports a case study of Design Space Exploration for supporting *Run-time Resource Management* (RRM). In particular the management of system resources for an MPSoC dedicated to multiple MPEG4 encoding is addressed in the context of an *Automotive Cognitive Safety System* (ACSS). The run-time management problem is defined as the minimization of the platform power consumption under resource and *Quality of Service* (QoS) constraints.

The Chapter provides an insight of both, design-time and run-time aspects of the problem. During the preliminary design-time Design Space Exploration (DSE) phase, the best configurations of run-time tunable parameters are statically identified for providing the best trade-offs in terms of run-time costs and application QoS. To speed up the optimization process without reducing the quality of final results, a multi-simulator framework is used for modeling platform performance.

At run-time, the RRM exploits the design-time DSE results for deciding an operating configuration to be loaded for each MPEG4 encoder. This operation is carried out dynamically, by following the QoS requirements of the specific use-case.

9.1 Introduction

The amount of resources available on MPSoC platforms is growing following the Moore's law. To fully exploit the potential capabilities of future MPSoC, programmers need to split applications in multiple threads in such a way that these can be allocated to different processors to be executed concurrently. There is not a unique way to parallelize an application but different parallel versions of the same application can be analyzed and used to trade off the application performance and the number of threads in the application. If every thread needs a dedicated processor to be executed, the availability of different parallelized versions can provide also

G. Mariani (✉)
ALaRI, University of Lugano, Switzerland
e-mail: giovanni.mariani@usi.ch

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_9, © Springer Science+Business Media, LLC 2011

a trade off between resources needed to execute the application and the application performance.

In this context, a *Run-time Resource Management layer* (RRM), part of the Operating System (OS), is expected to dynamically decide which application version has to be executed for each active application. In particular the design of future MPSoC should consider a complex multi-programmed scenario where many applications, each one composed of multiple parallel threads, are competing for accessing the shared computing resources.

The RRM problem becomes more complex as the number of run-time tunable parameters to be controlled by the OS increases; e.g., when operating frequencies of processors executing different threads have to be set together with the application parallelization. The RRM should set at run-time, for each active application, an operating configuration to dynamically match the user requirements while minimizing non-functional costs as energy or power consumption of the MPSoC platform.

Run-time decision making involves the solution of a combinatorial problem to fit, with the lowest cost and highest QoS, a set of predetermined resources. In fact, deciding which operating configuration has to be set for each application can be modeled as a Multi-dimension Multiple-choice Knapsack Problem (MMKP) [9] and belongs to the *NP-hard* class of problems [3].

Performing an exhaustive exploration of the possible candidate solutions at run-time would be too slow and even a trial-and-error procedure, where the RRM physically tries on the platform different operating configurations and then takes decisions based on observed system behaviors, would be unacceptable. To cope with this problem, in Chap. 5 we introduced a lightweight RRM exploiting design-time knowledge for efficiently selecting at run-time a reasonable assignment of the run-time tunable parameters while, in Chap. 6, we presented system-wide OS level methodologies that can be applied when detailed design-time information are not available.

The present Chapter demonstrates the effectiveness of the approach proposed in Chap. 5 for an industrial scenario. In this context, we target both the number of allocated cores and their operating frequencies as run-time configurable parameters of an application. Our case study concerns the automotive field and the RRM is responsible for allocating system resources to multiple instances of MPEG4 encoder where each instance has a different QoS requirement in terms of frame rate. The partitioning of system resources is here obtained by loading different parallel versions of the MPEG4 encoder to the different application instances. Each MPEG4 parallel version is composed of a different amount of threads and thus it needs for a different number of cores to be executed.

This Chapter shows both design-time and run-time sides of the methodology. During the design-time DSE phase, a multi-level simulation framework is used to search for the Pareto optimal operating configurations within the space identified by the run-time tunable parameters. Within the multi-level simulation framework a detailed but slow cycle accurate simulator and an approximate but fast High Level Simulator (HLSim) are available. This allows the quick exploration on HLSim of a large amount of the design space while the accuracy of the final solution is obtained

via validation on the cycle accurate simulator. This allows to speed up the exploration without reducing the quality of the final results.

The Chapter is organized as follows. In Sect. 9.2 we present the use case scenario introducing the MPEG4 application and its different parallel versions. Then, Sect. 9.3 gives an insight of the target MPSoC platform and the simulators used during the design-time DSE. Section 9.4 reports the results obtained from the proposed methodology during the design-time DSE phase first (Sect. 9.4.1) and, second, from the RRM during the run-time execution (Sect. 9.4.2). The Chapter finally concludes in Sect. 9.5 summarizing the most relevant results.

9.2 Case Study

The case study presented in this Chapter concerns the management of system resources for a multiple-stream MPEG4 encoding chip dedicated to *automotive cognitive safety* tasks. In particular we are targeting an MPEG4 encoding for a 4CIF video resolution. The application code is specifically optimized for compilation on the target MPSoC system in which the main computational element is the coarse-grain reconfigurable ADRES processor [5].

9.2.1 Application Description

The MPEG4 encoder is an industry-standard, block-based, hybrid video encoder. The structure of the MPEG4 encoder is shown in Fig. 9.1 and details can be found in [7]. Mainly the MPEG4 encoder is composed of the following functional blocks:

- **Motion Estimation (ME)** compares the current frame with a reference frame previously processed in order to estimate the motion within the frames.
- **Motion Compensation (MC)** compensates the estimated motion in the goal of increasing the efficiency of the compression.

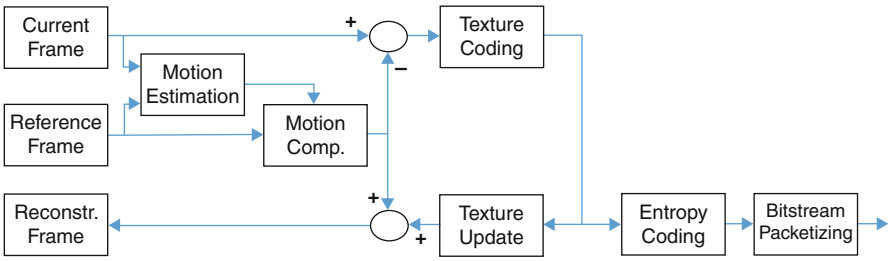

Fig. 9.1 Overview of the MPEG4 encoder application

- **Texture Coding (TC)** performs discrete cosine transform and quantization over motion compensated residuals ('texture').
- **Texture Update (TU)** uses the output of TC in order to locally reconstruct the original frame as it would appear after the decoding phase. This reconstructed frame can be useful later on as reference frame.
- **Entropy Coding (EC)** encodes the motion vectors to produce a compressed bitstream.
- **Bitstream Packetizing (BP)** prepares the packets containing the output data.

These functional blocks are implemented in application kernels (computational intensive nested loops) which have been optimized for compilation on VLIW architectures, in particular for the execution on an ADRES processor [5] used in our MPSoC platform.

The RRM can generate a trade off between application performance and resource usage by selecting a specific parallelization to be executed on the platform. To do so, the RRM needs different parallel versions of the same application, i.e., different binaries which perform the same functionalities while using a different amount of computing elements.

A set of parallel versions has been generated starting from a sequential implementation based on the MPEG4 Simple Profile reference code. First of all, the initial sequential version has been pruned and cleaned to set-up the parallelization procedure. Then, the sequential application is parallelized using *MPSoC Parallelization Assist* (MPA) tool [6].

MPA is a tool which supports MPSoC programmers on investigating different parallelization alternatives for a given application. Once the MPSoC programmer specifies a parallelization for the application, MPA is able to automatically insert into the sequential code all program lines needed to spawn parallel threads and to implement inter-thread communication. For generating different versions of the same application, the programmer should profile the sequential application, understand how kernels can be assigned to different threads and specify different parallelization opportunities to MPA, without any further handmade modification to the application code.

MPA is able to handle parallelizations either at *functional* or at *data* level (a combination of both functional and data parallelism is also handled). In practice, different functional kernels can be organized over different threads (functional parallelization) or the same kernel(s) can be divided over different threads by dividing them w.r.t. loop indices (data parallelization). In the second case, each thread performs the same functionalities over a different part of the dataset.

Once different parallel versions are generated with MPA, the obtained codes can be compiled and generated binaries can be executed on the target platform. In particular, Fig. 9.2 shows the parallel versions of the MPEG4 encoder studied during this Chapter. Within Fig. 9.2, the functional blocks are reported in solid boxes while the thread partitioning is represented by dotted lines.

In this case study, every thread needs a computing element to be executed and a computing element cannot execute more than one thread. Thus, the number of threads

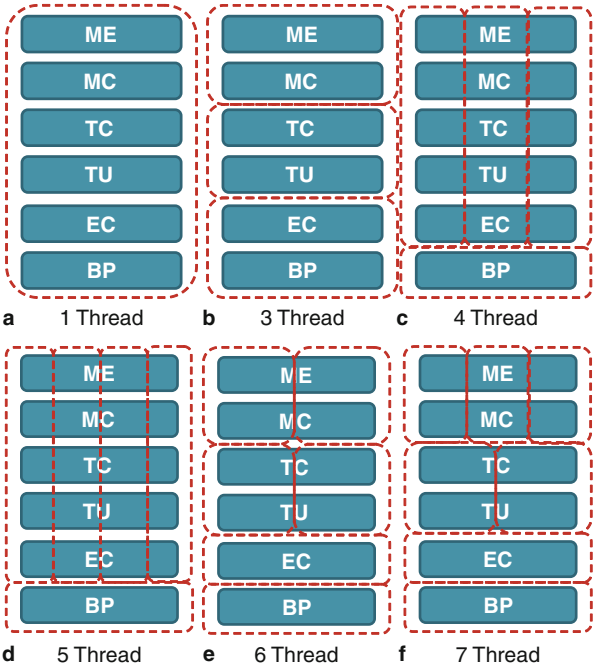

Fig. 9.2 MPEG4 encoder versions parallelized over a given number of threads (up to 7)

in an application version is equivalent to the resource cost ρ of the specific version. The six available MPEG4 encoder versions are parallelized for the execution on 1, 3, 4, 5, 6 and 7 processors. Since every version is parallelized over a different number of threads, in the following the resource cost ρ is also used to uniquely identify the MPEG4 encoder version.

9.2.2 Automotive Cognitive Safety System

The use case studied in this Chapter is an *Automotive Cognitive Safety System* (ACSS). In this context, a vehicle is equipped with a wide range of digital sensors (such as cameras and radars) and it is able to identify emergency conditions. The final goal of the ACSS is to keep passengers safer assisting the driver during emergency situations. In particular, the ACSS is able to identify and actuating emergency measures for the following scenarios:

- Forward collision warning.
- Automatic pre-crash emergency braking.
- Lane departure warning and guidance.
- Lane change and blind spot assist.

The vehicle is also provided with three digital cameras associated with left, right and central mirrors respectively, as shown in Fig. 9.3.

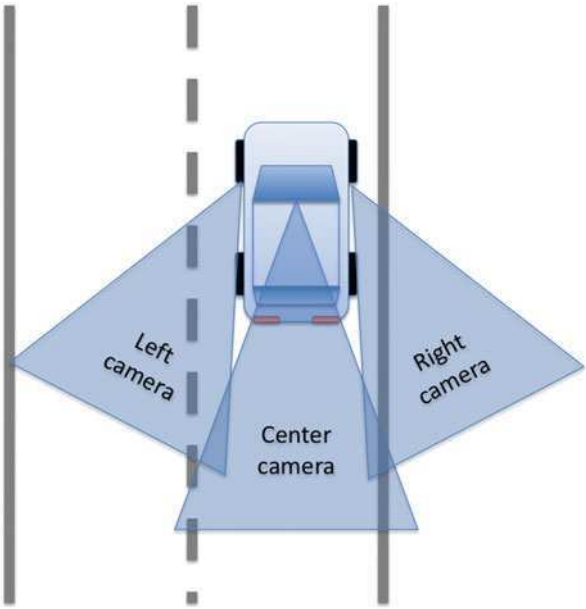

Fig. 9.3 Mirror views of the target vehicle

These cameras are connected to the target ADRES-based MPSoC platform which performs the encoding of the three video streams by means of the MPEG4 encoder presented in Sect. 9.2.1. For this purpose, an instance of the MPEG4 encoder is executed for each of the three views. The encoded streams are then sent to an off-chip Central Safety Unit (CSU), reducing the needs of on-board bus bandwidth.

Moreover, the dashboard has a set of displays that can be used to reproduce the actual content of the video streams sent to the CSU. Thus, the driver can watch live-views from the mirror cameras. The driver can independently switch on and off each of the camera views via the steering wheel interface.

We assume that the minimum requirement needed to let the CSU operating correctly is 15 frame per second (*fps*) per each video stream. Since this requirement is too low for providing the driver with a good video quality, the frame rate might be increased when the mirror views are displayed on the dashboard. In fact, when live-views are enabled, the CSU communicates new frame rate requirements to the MPSoC platform performing the video encoding.

In particular, when some live-views are activated, the requirements are decided on the basis of the vehicle speed and on the proximity of other vehicles. The following criteria are used:

- Lateral cameras: 15 *fps* under 10 km/h, 25 *fps* for over 110 km/h, interpolated linearly for intermediate speed. 30 *fps* when a vehicle is in proximity.
- Center camera: 15 *fps* under 20 km/h, 20 *fps* over 120 km/h, interpolated linearly for intermediate speed.

The frame rate requirements are not continuously updated but new requirements are communicated only when the vehicle speed passes certain thresholds or when another vehicle enters/exits the proximity areas. There is a threshold every 10 km/h for the lateral cameras and every 20 km/h for the central camera.

We here recall from Chap. 5 that the RRM needs also application priorities to decide which requirement should be relaxed when would be otherwise impossible to solve the RRM problem matching all requirements given the limited amount of system resources. The CSU communicates to the MPSoC platform also the priorities of the multiple video streams together with the frame rate requirements. By default the highest priorities are given to the central and left cameras. When a vehicle is in proximity, then the lateral camera closer to the approaching vehicle and the central camera get the highest priorities.

9.3 Platform Description

The industrial architecture studied is the 8-cores MPSoC shown in Fig. 9.4. The platform is composed of seven ADRES (coarse-grain Architecture for Dynamically Reconfigurable Embedded System) cores [5] and one StrongARM. The ADRES core is a power-efficient flexible computing element which combines a coarse-grain array with a VLIW DSP. The MPEG4 application has been optimized for compilation for the ADRES core in such a way that the data-flow loops can be efficiently accelerated by exploiting the coarse-grain array for loop level parallelization. The potentialities of the ADRES cores are fully exploited by executing the MPEG4 encoder instances while the RRM is executed on the StrongARM processor. The on-chip interconnect consists of a 32-bit wide Network-on-Chip with two switches.

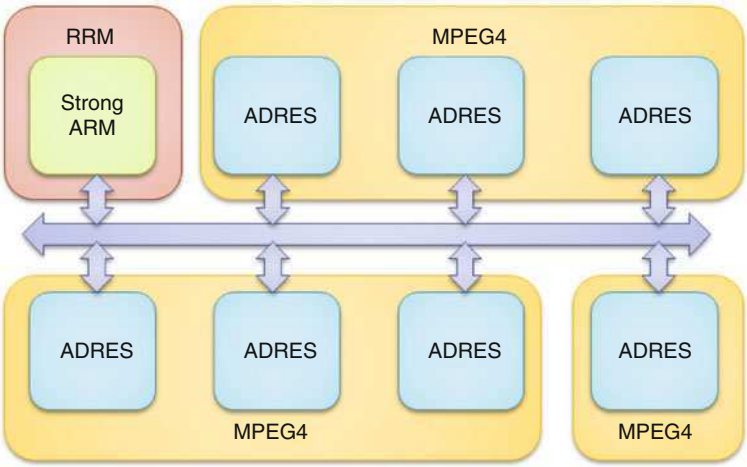

Fig. 9.4 Target MPSoC platform executing the RRM and the three MPEG4 encoder instances

The RRM can trade-off frame rate and power consumption of the multiple MPEG4 by acting on two parameters: the parallelizations of the MPEG4 instances and the frequencies of the ADRES cores. In this sense the operating frequency of each ADRES core can be dynamically modified independently by adopting *Dynamic Voltage and Frequency Scaling* (DVFS) techniques. In particular the frequency range for the ADRES cores is $\Phi = \{20, 60, 100, 140, 180, 220\}$ MHz, while the StrongARM is operating at 206 MHz.

9.3.1 Simulation Tool-Chain

For evaluating performance values of the MPSoC platform, a multi-simulator framework has been used [1]. More specifically, a *Transaction-Level Model simulator* (TLMsim) built with *CoWare platform* design tools enables the cycle-accurate analysis of the multi ADRES system and the underlying communication infrastructure.

This cycle-accurate simulator is too slow for enabling the analysis of many operating configurations for the MPEG4 encoder application. To cope with this problem a *High Level Simulator* (HLSim) has been used.

HLSim exploits back-annotated information of execution time derived from the cycle-accurate simulator. Execution time figures for all the kernels are recorded during cycle-accurate simulations and are fed back into HLSim as input while running the application.

In practice, during a first recording phase, the sequential version of the application is simulated on the cycle-accurate simulator; application kernels are profiled and performance indices (e.g. processor execution cycles) are saved into a database. Then, HLSim uses this profiled data library during its application simulation to derive accurate timing for application execution. HLSim does timed simulation of the application meaning that HLSim keeps track of local time in each thread and this time is appropriately adjusted during thread synchronizations. Performance indices are also scaled to the frequencies of the processors where kernels are executed. Given an application version and the frequency of each thread, the timing feedback mechanism let HLSim to provide a very fast but approximate evaluation of the platform performance indices.

Comparing HLSim with the cycle accurate TLMsim on simulating the MPEG4 encoding of 10 frames in 4CIF resolution, we obtain that HLSim has an execution time of 45 s with respect to TLMsim which requires 4 h. This simulation time saving is obtained at the cost of the simulation accuracy, in fact HLSim has a simulation error that is always lower than 20%.

The availability of HLSim enables the design-time DSE phase to quickly investigate many frequency combinations for each application version. The results obtained by HLSim for some simulations are then validated with the cycle accurate simulator. Cycle-accurate simulator can also be used for DSE phase but within a small focused interesting region obtained from HLSim-based DSE phase. This strategy of using two simulators enables application designers to evaluate DSE phase efficiently [1].

Finally, to complete the performance analysis of the simulation tool-chain, the performance figures of the StrongARM executing RRM are obtained using the *Simlt-ARM* simulator [8].

9.4 Experimental Results

This section presents the results obtained on applying the methodology proposed in Chap. 5 to the ACSS use case. In particular, we start presenting the results of the design-time DSE phase whose goal is the identification of the Pareto optimal operating points of the target application (Sect. 9.4.1). Then, behaviors of the proposed RRM on the specific case study are presented in Sect. 9.4.2.

9.4.1 Design Space Explorations Using Multi-Simulator Framework

In our use case, there are different versions of the MPEG4 encoder, each one identified by a specific resource cost ρ.

An operating configuration of the MPEG4 encoder is:

$$\boldsymbol{c} = \langle \rho, \phi, \pi, \tau \rangle \tag{9.1}$$

where ρ is the resource cost, ϕ is the frequency vector containing an operating frequency $\phi \in \Phi$ for each of the ρ threads, while π and τ are power and performance values obtained via simulation.

The design-time DSE phase has the goal of identifying the set of operating configurations $\boldsymbol{C}$ which are Pareto optimal considering resource cost, power consumption and average frame execution time (i.e., $\boldsymbol{c}[\rho]$, $\boldsymbol{c}[\pi]$ and $\boldsymbol{c}[\tau]$). During this design-time DSE phase, the frequency vector ϕ representing operating frequencies of platform cores is independently optimized for each MPEG4 version. In fact an independent optimization problem targeting minimization of power consumption $\boldsymbol{c}[\pi]$ and average frame execution time $\boldsymbol{c}[\tau]$ is solved for each version presented in Fig. 9.2. Results obtained from the different optimizations are then merged together and processed as explained in Chap. 5 to obtain the Pareto optimal operating points $\boldsymbol{C}$ and a user value $v(\boldsymbol{c})$ for each configurations $\boldsymbol{c} \in \boldsymbol{C}$.

The design space of each optimization problem grows exponentially with the number of threads ρ in the given application version. While the design space for the sequential version ($\rho = 1$) is composed of $|\Phi| = 6$ points only, the design space of the version parallelized over 7 threads is composed of $|\Phi|^7 = 279{,}936$ points. Given this difference in the design space dimensions, we use different algorithms to perform different optimizations (see Chap. 3). In particular all optimizations are performed with the Multicube Explorer tool [10]. Within this optimization environment we use a

Table 9.1 Design space size and simulations required for the optimization of each version of the MPEG4 encoder. In the last column, a summary of the overall design space and number of simulations needed for the optimization of all versions

ρ	1	3	4	5	6	7	Grand total
Design space	6	216	1,296	7,776	46,656	279,936	335,886
Simulations	6	216	309	567	694	1,077	2,869

full search optimization approach for the application versions where the design space is small enough (i.e., for $\rho \leq 3$); the Non-dominated Sorting Genetic Algorithm (NSGA-II) [2] is used when the design space grows too much (i.e., for $\rho > 3$). In particular the genetic algorithm is run with a population size $|\Phi| \times \rho$ for 50 generations.

Table 9.1 reports for each parallelized version of the application the design space size and the number of simulations needed to perform the optimization. Due to the huge number of simulations needed to perform the overall exploration (i.e., 2,869), only the *HLSim* is used within the optimization loop. Note that one simulation on HLSim takes around 45 s whereas the same simulation takes around 4 h on cycle-accurate simulator. Figure 9.5 shows the derived final Pareto configurations for the MPEG4 encoder encoding 10 frames at 4CIF resolution.

The *CoWare-based simulator* (TLMsim) is built to model a Transaction Level Model (TLM) of the platform at a *cycle-accurate level*. This simulator is used to validate the results obtained by HLSim and also this simulator is used to do a localized, focused explorations of the parameters not supported by HLSim. As explained

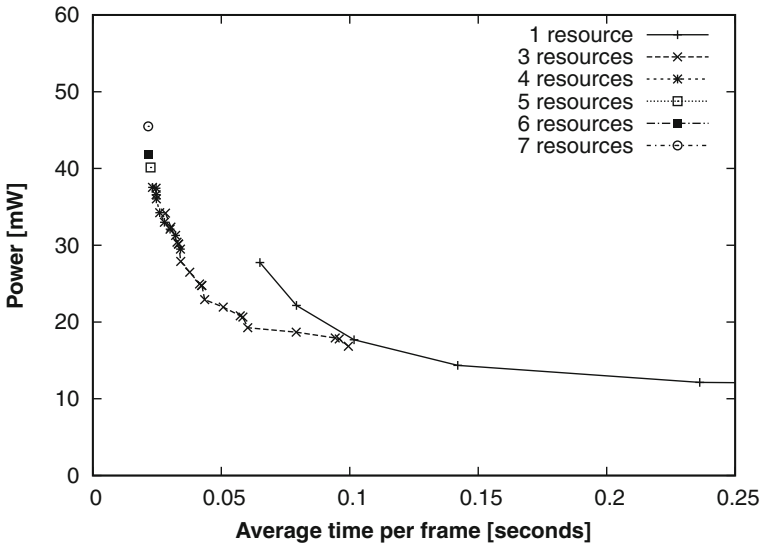

Fig. 9.5 Pareto optimal operating configurations for the MPEG4 encoder (encoding 10 frames at 4CIF resolution) by varying the number of resources

before, timing spent in different kernels during cycle-accurate simulations is profiled and fed back to HLSim. By this timing back-annotation, we remain accurate enough ($< 20\%$ error) between HLSim and cycle-accurate simulations w.r.t. execution time and power consumption. Further, Pareto operating points obtained by HLSim are also validated with cycle-accurate simulator. By using such a two-simulator approach, DSE phase can be done efficiently and accurately to reach optimum operating points in a short amount of time. Note that using such multi-level simulators to reduce number of simulations is orthogonal to sophisticated modeling techniques (e.g., Response Surface Modeling techniques presented in Chap. 4) supported by DSE tools [1].

To give an idea of the time saved by the proposed methodology we obtain that:

- The *exhaustive exploration* (i.e., the simulation of all 335886 configurations) with *TLMsim* would take 1,343,544 h; almost **153 years**.
- Performing the *NSGA-II optimizations* using *TLMsim* would require **16 months**.
- The *exhaustive exploration* with *HLSim* would take about **6 months**.
- Performing the *NSGA-II optimizations* using *HLSim* requires about **36 h**.

9.4.2 Run-Time Resource Management

9.4.2.1 Simulation of an Urban Environment

To generate dynamic QoS requirements for the RRM, the overall ACSS has been simulated for several reasonable driving patterns and conditions [4]. Although we simulated different patterns, for conciseness and clarity we will report results obtained for a specific pattern reproducing an urban scenario. The car speed and the frame rate requirements are shown in Fig. 9.6.

The vehicle, starting from a stationary position with speed equal to 0 km/h starts to accelerate a few before the 35th second. The vehicle reaches a maximum speed of 60 km/h around the 50th second and it keeps a constant speed for a while. Then, at about the 60th second, the vehicle starts decelerating to reach again a stationary condition at about the 70th second (Fig. 9.6a). All three live-views are active on the dashboard at the beginning of the simulation. The live-view of the central camera is deactivated from the driver at around 50 s and the frame rate of the corresponding video stream is reduced to 15 *fps* as required from the CSU (Fig. 9.6c). During the simulation two vehicles pass within the proximity area on the left side. The frame rate requirement of the corresponding camera increases to 30 *fps* (Fig. 9.6b).

9.4.2.2 Run-Time Resource Management Behaviors

Whenever the CSU sets new frame rate requirements, the RRM is invoked to globally select one operating configuration for each MPEG4 encoder with the goal of fitting

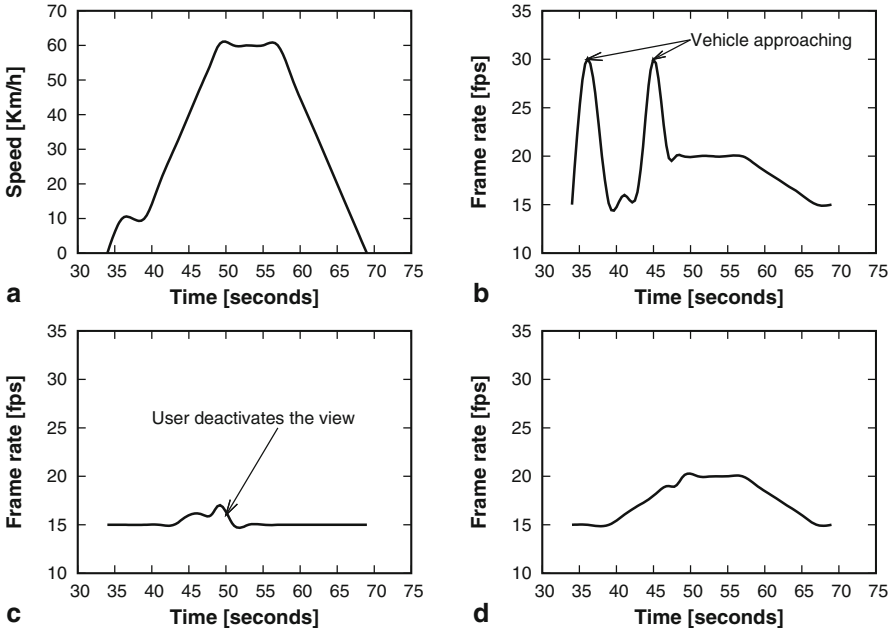

Fig. 9.6 Car speed and QoS for the urban case-study. **a** Car Speed, **b** Left camera, **c** Center camera, **d** Right camera

the QoS requirements while minimizing the power consumption without exceeding the amount of computational resources physically available on the MPSoC platform.

Figure 9.7 shows the selected MPEG4 encoder configurations in terms of number of allocated ADRESs and their average operating frequency set by the RRM for the three video streams given the driving pattern presented in Fig. 9.6.

Figure 9.8 reports also the power consumption of the overall MPSoC platform together with the penalties on the video stream requirements (i.e., the difference between the required frame rates and the achieved ones).

When the frame rate requirement is 15 *fps*, the RRM can set either a three-resources low frequency configuration or a one-resource high frequency configuration. In a normal stationary situation, the RRM provides one resource to the left camera and three resources to the central and right camera.

When a vehicle is in proximity of the left camera, the QoS requirement of this camera is of 30 *fps*. To fit in this requirement, the RRM has to set the left camera MPEG4 encoder in an operating configuration with many cores operating at high frequency. For these cases, required resources are taken from other video-streams which will move to a configuration with low resources and high frequency. In the overall system there will be many cores operating at high frequency and these situations are clearly characterized by peaks in the power consumption as visible in Fig. 9.8a.

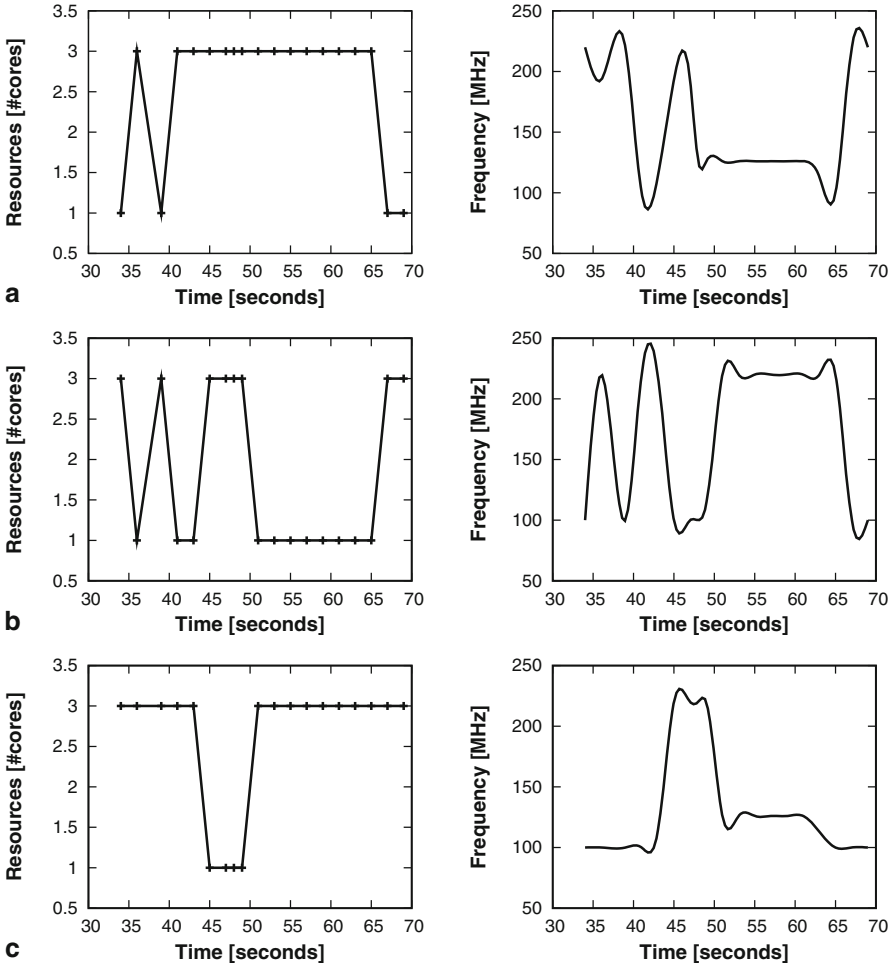

Fig. 9.7 Resource allocation and average operating frequency for the three MPEG4 instances. **a** Left camera, **b** Center camera, **c** Right camera

As the car speed increases, the frame rate requirements for the lateral cameras increase faster than the requirement for the central camera (Fig. 9.6). This brings around the 40th second to a situation where all requirements can still be met reducing the resources for the central camera while providing three cores to each lateral camera. This situation stands until the requirements cannot be met anymore. In this last situation, the requirement of the lowest priority application, i.e., the MPEG4 encoder of the right camera, is relaxed and only one resource is assigned to it. This situation is also visible in Fig. 9.8b, where the right camera frame rate is lower than the QoS requirement by around 4 *fps* (QoS penalty of -4 *fps*).

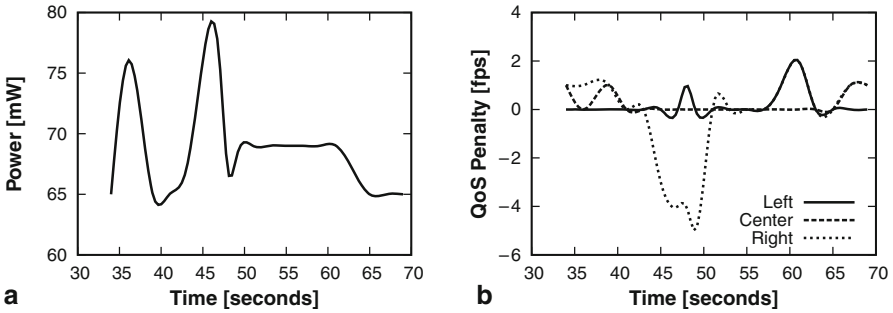

Fig. 9.8 System power profile and QoS penalty for the three MPEG4 instances. **a** Power, **b** QoS Penalty

During the simulations, the RRM routine (executed on StrongARM running at 206 MHz) for the operating point selection always found a solution within a computation time of 1 ms. This run-time overhead is enough small to be considered negligible in the given context.

To further clarify the advantage given by the availability of different parallelized versions of the same application, Fig. 9.9 shows a *bubble plot* representing the operating configurations effectively used by the RRM within the proposed urban scenario. A bubble plot is a way of representing the relationship between three or four variables on a scatter-plot. Observations on two variables are plotted in the usual way on the x and y axis using circles as symbols. The radii of the circles are made proportional to the associated values for the third variable while different gray levels are used to represent the fourth variable. In our specific case, the average operating frequency and the power consumption of the operating configurations are respectively reported on the x and y axis; frame rate supported from the configurations is proportional to the circles dimensions while the gray level represents the amount of ADRES cores needed to execute the configuration.

It is worth to note that among the configurations loaded by the RRM, only one uses the sequential version of the MPEG4 encoder (i.e., the version running on one

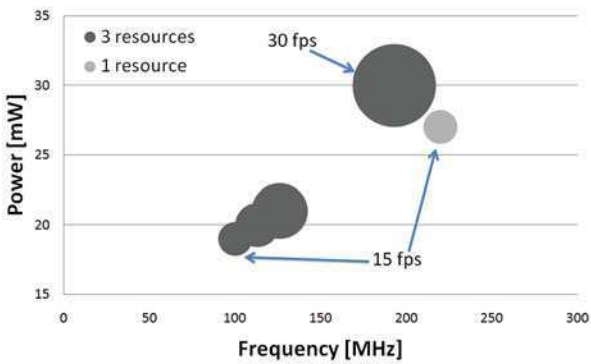

Fig. 9.9 Bubble plot of operating points loaded by the Run-time Resource Management

single ADRES core). This particular configuration executes the MPEG4 encoder at 220 MHz and provides a frame rate of 15 *fps*.

The MPSoC platform with seven ADRES cores is over-dimensioned for the minimal requirement of 15 *fps* for each video stream needed by the CSU to operate correctly. In fact for executing 3 MPEG4 encoders at 15 *fpss*, 3 ADRES cores would be enough. The availability of different parallel versions provides to the RRM the possibility to allocate the remaining resources to minimize the platform power consumption. In particular, the RRM under the minimal requirements steadily selects a configuration with 3 resources for the right and central cameras. The RRM takes this decision since the MPEG4 encoder running on 3 cores can be executed at about 100 MHz, obtaining the same QoS and saving more than 5 mW of power consumption (per 10 frames).

We can also observe that whenever a solution which matches all QoS requirements exists, this is correctly identified by the priority-based RRM. In fact, from Figs. 9.8b and 9.6 it is possible to notice that there is a QoS penalty only when all cameras require more than 15 *fps*. In fact, providing more than 15 *fps* requires as least three ADRES cores. Thus, due to resource constraints, when all video streams require more than 15 *fps*, the frame rate of the lowest priority stream has to be relaxed.

9.5 Conclusions

In this Chapter we presented an application of the RRM to the management of resources dedicated to a multiple-stream MPEG4 encoder chip within a *Automotive Cognitive Safety System* scenario.

First of all the MPEG4 encoder application has been presented together with the specific use case requirements. The design space identified by the run-time tunable parameters has been considered to be too huge to be explored exhaustively during the design time DSE phase. This is the reason for performing a heuristic optimization within an integrated DSE framework which provides optimization algorithms and RSM techniques as the ones described in Chaps. 3 and 4.

For enabling this design-time optimization phase, a multi-simulator framework was adopted. We showed that a *High Level Simulator* (HLSim) can be used to quickly evaluate performance indices of many candidate operating configurations. These obtained performance results are validated with cycle accurate simulations.

Once the Pareto optimal operating configurations are identified, the RRM previously proposed in Chap. 5 has been proven to be effective at identifying an operating configuration for each of the multiple video stream. In the experiments, the RRM always identified a configuration which is able to satisfy all constraints when this solution exists. When the QoS requirements are too high to be matched with the available resources, the priority based algorithm can still identify a solution to the problem by relaxing the constraint on the application having the lowest priority.

Note that the techniques for run-time resource management discussed in this Chapter are valid when ample design time information of all the running applications

is available. When such a design-time information of different applications is not available, system wide approaches for run-time management at the Operating System (OS) level can be applied. Such OS level approaches are discussed in Chap. 6.

References

1. Avasare, P., Vanmeerbeeck, G., Kavka, C., Mariani, G.: Practical approach for design space explorations using simulators at multiple abstraction levels. In: Design Automation Conference (DAC) Users' Track (2010)
2. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. Proceedings of the Parallel Problem Solving from Nature VI Conference pp. 849–858 (2000). URL citeseer.ist.psu.edu/article/deb00fast.html
3. Garey, M.R., Johnson, D.S.: Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman (1979)
4. Mariani, G., Avasare, P., Vanmeerbeeck, G., Ykman-Couvreur, C., Palermo, G., Silvano, C., Zaccaria, V.: An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In: DATE 2010 - International Conference on Design, Automation and Test in Europe., pp. 196 –201. Dresden, Germany (2010)
5. Mei, B., Sutter, B., Aa, T., Wouters, M., Kanstein, A., Dupont, S.: Implementation of a coarse-grained reconfigurable media processor for avc decoder. Journal of Signal Processing Systems **51**(3), 225–243 (2008). DOI http://dx.doi.org/10.1007/s11265-007-0152-8
6. Mignolet, J.Y., Baert, R., Ashby, T.J., Avasare, P., Jang, H.O., Son, J.C.: Mpa: Parallelizing an application onto a multicore platform made easy. IEEE Micro **29**, 31–39 (2009). DOI http://doi.ieeecomputersociety.org/10.1109/MM.2009.46
7. Richardson, I.: H. 264 and Mpeg-4 Video Compression: Video Coding for Next-generation Multimedia. John Wiley & Sons (2003)
8. Simlt-arm (2007). Http://simit-arm.sourceforge.net/
9. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In: Proceedings of International Symposium on System-on-Chip, pp. 1–4 (2006). DOI 10.1109/ISSOC.2006.321966
10. Zaccaria, V., Palermo, G., Mariani, G.: Multicube explorer (2008). Http://www.multicube.eu

Conclusions

With our work on MULTICUBE project and this book, we have pushed towards the adoption of automatic design space exploration for the design of multi-processor architectures for embedded computing systems. To demonstrate the effectiveness of the design techniques developed in the MULTICUBE project, besides the results already reported in Part II of the book, several assessment and validation activities have been carried out on real industrial design vehicles defined as use cases. The assessment has mainly been based on the application of the automatic DSE assessment procedure as defined in Chap. 1. The procedure has then been adapted by the responsible of each use case to comply with the specificities of its particular application and context, but without modifying the aim and the general steps of the original one. The main results of the comparison of the application of a manual or semi-automatic optimisation flow with respect to an automatic optimisation flow can be summarized by the following considerations.

DS2, the leading supplier of silicon and software for Powerline Communications (recently DS2 assets and technologies were bought by Marvell Technology Group) has applied the procedure to its Powerline use case described in Chap. 7 and suitable for advanced communication system to enable a fast and reliable transfer of audio, video and data information by using the most ubiquitous transmission system: the power lines. The general DSE assessment procedure has then be adapted to the peculiarities of the Powerline use case to test how the different methodologies and tools developed in the MULTICUBE project can be effectively applied to this industrial design flow. The advantages found by DS2 on the target use case can be summarized as follows:

> The proposed flow can save up to 80% of designer time while achieving better results in terms of performance since much more simulations can be run and analyzed by following this flow obtaining ten times more possible combinations than with the semi-automatic design flow.

STMicroelectronics, one of the world's largest semiconductor companies, has applied the MULTICUBE design flow to design and optimize the SP2 low power advanced computing processor developed by STM Beijing for generic computing applications. The design space of the use case is very large including several architectural parameters concerning out-of-order execution parameters, memory hierarchy parameters and branch prediction parameters. To quantify the benefits of using the

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9, © Springer Science+Business Media, LLC 2011

MULTICUBE design flow in an industrial practice, STM compared the traditional optimization procedure used internally with respect to the automatic DSE procedure proposed by MULTICUBE project. The benefits found from STM can be summarized by the following statement:

> The use of MULTICUBE optimization flow can save up to a 73% of the overall time while achieving comparable results in terms of power/performance tradeoffs . . . . The overall conclusion of the utilization of MULTICUBE design flow over a real industrial use case like the one just discussed is that the added value overcomes largely the cost of setting up the whole process, converging much faster to the optimum solution for a given technical problem.

IMEC, the Europe's leading independent research center for the development of microelectronics and ICT, was interested in evaluating the MULTICUBE DSE flow to show proof-of-concept of its innovative research ideas. IMEC was interested in evaluating the benefits of DSE flow not only considering the design-time tunable parameters, but also using the exploitation of the approach at run-time. For proof-of-concept of both design-time and run-time automatic DSE, IMEC used the same MPSoC virtual platform based on ADRES core executing the MPEG4 encoder application described in Chap. 9. The benefits of automated DSE with respect to the manual approach can be summarized by the following statements:

> Looking at the two design space exploration case-studies, it can be seen that using automatic DSE over manual full-space exploration has large benefits in terms of time-to-market and accuracy of exploration results. By using automatic DSE in dynamic runtime management evaluation case-study, design space exploration time was reduced from 6 months (full-space) to 36 h. In another case-study of MPSoC platform optimization, by using a simulator at higher abstraction level coupled with automatic DSE, we could reduce design space exploration time from worst case of 153 years (full-space) to 36 h . . . . The extra efforts required to build automatic DSE procedure were minimal.

Besides these tangible benefits of using automated design exploration techniques obtained by implementing a number of industrial use-cases, MULTICUBE project demonstrated several achievements at the scientific/technical level. Based on the opinion of the European reviewers of the project, these achievements can be summarized by the following statements:

> . . . Practical ways to trade off accuracy for speed through the use of multi-abstraction level simulation thus enabling exploration of larger design spaces.
> . . . The feasibility of automated parameter tuning at run-time using exploration data collected at design-time. By implementing and applying a number of analytical and computational multi-objective optimisation techniques the project has demonstrated the advantages (in terms of design costs) of using automated architectural exploration at design-time and also demonstrated the possibility of performance improvements for run-time.
> . . . The effectiveness of using a common XML scheme to enable integration and interoperability of both propriety and open-source software tools in a coherent design tool-chain.

To conclude, in the age of multi/many-core architectures, we believe that system optimization and exploration technologies and tools represent not only a challenging research task to be further investigated in the next coming years, but could also represent one of the potential research breakthroughs.

The Editors

Index of Terms

A
ADRES, 7, 8, 191, 192, 194–196, 200, 202, 203, 206
Application version, 102, 190, 193, 196–198
Application working mode (AWM), 135, 136
Automatic design space exploration, 5, 6, 13, 14, 52, 172
Automatic exploration, 14–16, 180
Automatic model generation, 22, 23
Automotive Cognitive Safety System, 189, 191, 193, 203
Auto-scaling, 83
Average normalized error, 91, 185

B
Benchmark problem, 66, 69, 70, 72
Box-Cox power transform, 83

C
Cache modeling, 21, 32, 33, 47
Categorical discrete optimization, 176
Categorical variables, 13, 52, 54, 55, 58–60, 64, 72, 80, 81
Central composite design of experiments, 79
Closed-form analytical expressions, 75, 76, 91
Co-design hardware/software, 114, 139
Core, 8, 29, 33, 53, 75, 76
Cross-layer RTRM techniques, 133
  Constrained Power Management (CPM), 134
  Dynamic Power Management (DPM), 131, 132
  QoS-PM Framework, 133
Curve fitting, 78
Cycle-accurate system-level simulation, 76
  Transaction-Level Model (TLM), 8, 40, 98, 196, 198

D
Data distribution, 83
Design flow, 3–5, 8, 10, 11, 17, 23, 84, 139, 168–170, 171
Design flow optimization, 146, 158, 163, 206
Design of Experiments, 6, 9, 52, 53, 57, 75, 77, 79, 160, 175
Design space definition, 11, 12, 174, 182
Design space exploration, 3–6, 9, 10, 13–15, 17, 19, 20, 47, 51–53, 55, 62, 66, 70, 73, 75, 76, 91, 93–95, 98, 105, 136, 145, 146, 171, 172, 174, 188, 189, 197
Design time heuristics, 53, 96, 97
Design tools, 4, 5, 11, 84, 174, 196
Designer time reduction, 168, 169
Design-time Design Space Exploration, 95, 189
Discrete variables, 57, 80–82
  Dynamic Power Management (DPM), 132
Dynamic Voltage and Frequency Scaling, 96

E
Early-stop criterion, 80
Elitism, 56–58, 63, 64, 71
Embedded computing systems, 80
Embedded multimedia platforms, 4
Energy estimation, 29
Event-based simulator, 75, 76
Evolutionary Design, 84, 90, 185, 187

F
Feed-forward Neural Network, 87, 88
Full factorial design of experiments, 79
Functional-level simulation, 98

G
Gaussian process, 88
Genetic algorithms, 7, 53, 55, 57, 63, 102, 198
Genetic programming, 90

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9, © Springer Science+Business Media, LLC 2011

H
Hidden layer, 87, 88
High-level modeling, 145
HLSim, 8, 9, 98, 190, 196, 198, 199, 203

I
Industrial Design Space Exploration Data Mining, 6
Interpolation, 9, 78, 84, 85, 89, 90

K
Kriging, 84, 88, 89, 184–186

L
Leave-one-out predictive probability, 89
Linear regression, 9, 84, 85, 184–186
Low power processor, 66, 171, 172, 188

M
Many-core architectures response surface modeling, 172, 188
Memory Subsystem optimization, 112
Mixed-workload, 137
Model selection, 15, 78, 80, 85
Model training, 75, 83
Modeling tools, 3, 5–7, 17, 39, 45, 98, 102, 154
MPEG4, 8, 189–203
Multi/many-core architectures, 4, 8, 109, 113, 171, 172, 188, 206
Multi-dimension Multiple-choice Knapsack Problem, 95, 97, 99, 101, 104, 190
Multi-layer perceptrons, 87
Multi-objective optimization, 52, 53, 55, 58, 60, 95, 103, 134–136, 174
Multi-processor SoC architectures, 5

N
Native co-simulation, 20, 21, 23, 36, 37, 46, 47
Network topology, 156
Network-on-Chip, 8, 182, 195
Neural Networks, 9, 84, 87, 185, 186
Noise parameters, 88
Non-dominated Sorting Genetic Algorithm, 55, 56, 59, 60, 63, 102, 198, 199
Numerical stability, 81, 82

O
Open-source design tools, 3, 5, 7, 10
Open-source frameworks, 3, 5, 7
Operating point (OP), 8, 10, 93, 97, 99–104, 116, 132, 133, 139, 197, 199, 202
  OP selection, 116, 132, 133
Operating point of an application, 100
Operating system, 7, 10, 23, 24, 36–38, 95, 105, 109, 115, 117, 124, 128, 134, 136–139, 149, 190, 204
Operating system support, 117
Optimization workflow, 174
Optimization (multi-objective optimization), 52, 53, 55, 60, 95, 134–136, 174
Optimization algorithms, 16, 51–53, 55, 72, 73, 105, 172, 180, 203
OS mechanisms, 119
Over-fitting, 78, 80, 85, 87

P
Paramerizable architectures, 86, 101
Pareto domination, 63
Pareto front (Pareto set), 7, 17, 51, 53, 55, 56, 57, 63, 64, 66–70, 72, 176, 177, 179
Pareto set, 3, 4, 8, 10, 58, 60, 65, 70, 72, 95, 98
Performance evaluation, 24
Performance metric, 19, 86
Piecewise polynomials, 86
Platform-based design, 7
Polynomial, 78, 85–87
Post Processing, 52, 53, 72
Powerline-communication network, 8, 145–147, 150
Prediction error, 88
Processor architectures, 75, 76, 180, 181
  QoS Estimator, 116
  QoS-PM Framework, 133

Q
Quality of service requirement, 131
Quality-of-Service (QoS), 4, 8, 10, 93, 94, 97–99, 101, 103, 105, 112, 116, 117, 131, 133, 134, 136, 139, 145, 146, 156, 189, 190, 199, 200, 201, 203

R
Radial Basis Functions, 9, 81, 85, 184–186
Random design of experiments, 53, 79, 82
Regression, 9, 78, 84–86, 88, 90, 91, 184–186
Response surface model, 9, 75–78, 91, 172, 180, 184, 199
Response surface modeling techniques, 9, 179
Restricted cubic splines, 87
RSM, *see* Response surface model
RTOS modeling, 33
RTRM Linux Frameworks, 120, 130
  Clock, 76, 95, 116, 124, 125, 128, 129
  CPUFreq, 126, 127
  CPUIdle, 125, 134
  Suspend and Resume, 128
  Voltage and Current Control, 130

Run-time Design Space Exploration, 4, 117, 190
Run-time resource management (RTRM), 112
  Adaptive RTRM, 114
  Centralized RTRM, 114, 131, 139
  Distributed RTRM, 114, 131, 139

S
Simulation speed, 21, 22, 75, 76, 173
Simulation Tools, 3, 5, 7, 14, 17, 20, 22, 25, 196, 197
Simulator, 4–13, 16, 25, 28, 41, 42, 45, 52, 54, 66, 67, 72, 73, 75, 76, 93, 96, 97, 101, 103, 105, 169, 173–175, 182, 183, 189–191, 196–199, 203
Soft real-time system, 101
Splines, 84, 86, 87, 184–186
Statistical sampling, 76
Steady state evolution, 59, 64, 72
Symbolic regression, 90
SystemC, 4, 7, 19, 20, 22–24, 28, 34, 35, 39, 41, 43, 45, 47, 98, 145, 148–151, 154
System-level Design, 22
System-level metrics, 8, 75, 77
System-level requirements, 114, 118, 125, 127, 128, 134, 138
System-level specification, 17
System-on-Chip architectures, 3
System-wide optimization, 114, 128, 131, 134, 138, 139

T
Time approximate, 4, 7, 190
Timing-annotated functional level simulation, 8, 21, 22, 196
TLMsim, 98, 196, 198, 199
Tool integration, 4, 11
Traffic shape, 149, 156
Training algorithm, 80, 81, 88
Training set, 10, 77, 78, 80, 86, 91, 185, 187
Transaction Level Modeling, 22, 33, 149, *see also* TLM

U
Uniform Latin Hypercube, 82, 176

V
Validation, 5, 8, 16, 53, 66, 71–73, 77, 80, 83, 91, 184–186, 191
Validation set, 80
Variance, 69, 83, 85, 185
Variogram, 89, 185
Versioning, 99
Virtual platforms, 14, 38, 98, 145, 146, 148–150, 152, 154, 163, 206
VLIW processors, 8

X
XML interface, 11–13, 41